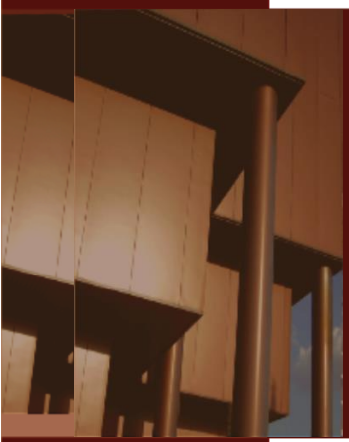
Implementación y análisis de un sistema de adquisición de datos para un vehículo ferroviario a escala

Autor: María Dolores Sancho Martín

Tutor: Daniel García Vallejo



Sevilla, 2018-2019

UNIVERSIDAD DE SEVILLA

tsc
Departamento de Teoría de la Señal y Comunicaciones

Trabajo Fin de Máster
Máster Universitario en Ingeniería de Telecomunicación

Implementación y análisis de un sistema de adquisición de datos para un vehículo ferroviario a escala

Autor:
María Dolores Sancho Martín

Tutor:
Daniel García Vallejo
Profesor titular

Dep. de Ingeniería Mecánica y Fabricación
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018-2019

Agradecimientos

En primer lugar, agradecer a mis padres, mi hermano, abuelos y toda mi familia el apoyo incondicional junto con las palabras de aliento necesarias para alcanzar las metas que me he propuesto a lo largo de mi vida. También a mi novio por haber sido y ser un gran pilar para llevar a cabo estos años de duro trabajo.

A los compañeros del departamento de Ingeniería Mecánica y Fabricación, por darme la oportunidad de formar parte de su equipo abriéndome las puertas del ámbito laboral, así como todo el conocimiento y confianza aportados.

Finalmente, a mi tutor Daniel García Vallejo por todo su apoyo y ayuda en la realización de este proyecto.

Resumen

El progreso tecnológico de los medios de transporte actuales es esencial para la evolución de una sociedad donde se apuesta por la interconexión total, tanto utilizando los medios de comunicación como en la realización de desplazamientos gracias a los medios de transporte.

Para conseguir esta mejora, primero es necesario un estudio de la situación existente. Por este hecho, se estudian las capacidades de los vehículos ante el deterioro por la exposición a perturbaciones durante el ciclo de vida del vehículo.

Con dicha motivación, se realiza el desarrollo de un software de gestión de datos obtenidos a partir de un sistema de adquisición, consistente en una red cableada de sensores, desplegado sobre un vehículo ferroviario a escala.

El proyecto expuesto en el presente documento muestra un análisis completo del sistema de adquisición de datos ubicado sobre el tren a escala del departamento de Ingeniería Mecánica y Fabricación de la Escuela Superior de Ingeniería de la Universidad de Sevilla. De este modo, se muestra la estructura de dicho sistema, el flujo de datos sobre la misma, el procesamiento de las diferentes señales que contienen los datos obtenidos por los sensores y la automatización del post-procesamiento de dichos datos, con el propósito de capacitar al sistema de la realización de representaciones de las muestras de los ensayos de forma inmediata en campo.

Abstract

The technological progress of the current transport is essential for the evolution of a society where it is committed to total interconnection, both in media and in means of transport.

In order to achieve this improvement, a study of the existing situation is needed. For this reason, the capacities of the vehicles are studied before the deterioration due to exposure to disturbances during the life cycle of the vehicle.

With this motivation, it is carried out the development of a data management software obtained from an acquisition system, distributed by a railroad vehicle at scale, consisting of a wired sensor network.

The project presented in this document shows a complete analysis of the data acquisition system located on the scale train of the Department of Mechanical Engineering and Manufacturing of the Higher School of Engineering of the University of Seville. In this way it are shown, the structure of said system, the flow of data on it, the processing of the different signals containing the data obtained by the sensors and the automation of the post-processing, with the purpose of training to the system of performing representations of test samples immediately.

Índice

ÍNDICE DE FIGURAS

Notación

≤	Menor o igual
≥	Mayor o igual
~=	Distinto de
Input	Señal/variable/archivo de entrada para un dispositivo HW o algoritmo SW
Output	Señal/variable/archivo de salida para un dispositivo HW o algoritmo SW
HW	Hardware
SW	Software
UART	Universal Asynchronous Receiver-Transmitter
IMU	Inertial Measurement Unit
optoNCDT	Optical Non-Contact Displacement Transducer
ESI	Escuela Superior de Ingeniería
US	Universidad de Servilla
PATH	Ruta de directorios donde se encuentra el fichero deseado
SAD	Sistema de Adquisición de Datos
mm	milímetro
AS-IS	Escenario inicial o actual
TO-BE	Escenario futuro, al que se desea evolucionar
FPS	Fotogramas por segundo

1 INTRODUCCIÓN

El proyecto que se presenta a continuación muestra un análisis completo del sistema de adquisición de datos ubicado sobre el vehículo ferroviario a escala, el cual es objeto de estudio del departamento de Ingeniería Mecánica y Fabricación de la Escuela Superior de Ingeniería de la Universidad de Sevilla. Con la instalación de dicha instrumentación, se capacita al vehículo para que realice la auscultación de las vías ferroviarias sobre las que se desplaza y del estudio de la dinámica vehicular ante las perturbaciones que presente las vías.

El desarrollo de este proyecto se puede dividir en dos grandes bloques, siendo el primero un estudio de la situación inicial en la que se encontraba el vehículo, al comenzar la realización de este proyecto, y el segundo el desarrollo de un sistema de análisis de datos obtenidos de los ensayos realizados al vehículo instrumentado. La inclusión del primer bloque se considera pertinente ya que, según define la metodología de gestión de proyectos, primero hay que analizar de la situación actual para poder realizar un desarrollo adaptado a ella. Así mismo, en el segundo bloque se detalla el procesamiento de las diferentes señales que contienen los datos obtenidos por los sensores y la automatización del post-procesamiento de dichos datos. Dicha automatización tiene el propósito de capacitar al sistema de la realización de representaciones de las muestras tomadas en los ensayos de forma inmediata en campo, de este modo se puede comprobar si la toma de datos del ensayo ha transcurrido de forma correcta y se han obtenido datos relevantes para su análisis.

Gracias a los proyectos realizados previamente a este trabajo en el departamento de Mecánica y Fabricación de la ESI, como es el creado por Juan Manuel Amador y tutorizado por José Luis Escalona [1], tanto el sistema de adquisición de datos como el propio vehículo ferroviario a escala se encontraban en funcionamiento y en periodo de ejecución de ensayos.

1.1 Objetivo

El objetivo de este proyecto es el desarrollo de un software de gestión de datos obtenidos a partir de un sistema de adquisición consistente en una red cableada de sensores y distribuida por un vehículo ferroviario a escala. El trabajo incluye el procesamiento de las diferentes señales y la automatización de la ejecución del mismo, para poder realizar su visualización inmediata al realizar las pruebas en campo.

1.2 Estructura del proyecto

Esta memoria se estructura siguiendo la metodología de gestión de proyectos constituyente en cinco fases principales: estudio de la situación inicial, análisis y toma de requerimientos para la solución, definición y planificación del proyecto, desarrollo de la solución y, finalmente, ejecución y testeo de la solución.

Como se ha comentado anteriormente, este proyecto se puede dividir en dos grandes bloques, siendo el primero el estudio del AS-IS del vehículo ferroviario a escala instrumentado, y el segundo el desarrollo del software de gestión y análisis de datos, objeto de este proyecto. Adaptando la metodología de la gestión de proyectos a este proyecto, se pueden definir cuatro grandes paquetes de trabajo, los cuales pueden definirse como:

1. Estudio AS-IS

   Se comenzará exponiendo la estructura inicial que presenta el vehículo a escala instrumentado, incluyendo tanto la descripción, distribución y conexión de los diferentes componentes de la red cableada, como la lectura y procesamiento de las señales procedentes de los sensores.

2. Análisis y toma de requerimientos

   En esta etapa se realizan tres análisis fundamentales, siendo el primero la representación de los datos obtenidos por los sensores, el segundo la resolución de datos espúreos presentados en la lectura de los sensores en la solución inicial y el tercero la posibilidad de automatización del post-procesado de los datos.

3. Desarrollo de la solución

   Dentro de la solución, se contemplan tres grandes bloques, comenzando con la representación de los datos y continuando con la resolución de los datos espúreos, lo cual implica una mejora en la sincronización de los sensores y procesamiento de la señal procedente de los mismos. Como tercer bloque, se lleva a cabo uno de los bloques principales de este proyecto, el desarrollo de la automatización del post-procesado de las señales leídas por los sensores para poder realizar una representación de los datos inmediatamente después de realizar los pruebas en campo y comprobar la validación de los resultados para su estudio.

4. Ejecución y testeo de la solución

   Esta última sección se presenta como comprobación del desarrollo expuesto en el capítulo anterior, representando una muestra de los resultados que aporta el sistema de adquisición en cuatro ensayos realizados en campo.

2 ANÁLISIS ESTRUCTURA AS-IS

Se comienza con este capítulo con el análisis de la situación AS-IS, es decir, la situación inicial en la que se encontraba el vehículo ferroviario instrumentado al comenzar la realización de este trabajo, cuyo desarrollo fue realizado por el departamento de Ingeniería Mecánica y Fabricación.

Este estudio se presenta dividido en dos capítulos, siendo este el primero donde se muestra la estructura hardware de los diferentes equipos que componen el sistema de adquisición de datos, así como se indica su localización en el vehículo ferroviario y las conexiones que presentan entre ellos. Ambos capítulos se basan en la información expuesta en el trabajo fin de grado de Juan Manuel Amador, tutorizado por José Luis Escalona, el cual proporciona la información detallada del sistema al ser objeto de desarrollo de dicho proyecto, denominado *Instrumentación de un vehículo ferroviario para la auscultación dinámica con sistemas de visión artificial*, como puede verse referenciado a lo largo de los dos capítulos.

Por su parte, se profundiza en los diferentes sensores que componen la red de adquisición de datos, tanto los encargados de la auscultación de la vía como los encargados de la adquisición de datos de la mecánica vehicular. Los primeros son dos cámaras de vídeo que enfocan cada una a un láser de proyección lineal el cual se ha proyectado sobre cada una de las dos vías ferroviarias. Adicionalmente, para la auscultación de la vía, una tercera cámara es la encargada de grabar todo el recorrido del tren a su paso por la vía en los ensayos. Sin embargo, esta última cámara no se ha considerado oportuno profundizar en ella ya que no está conectada directamente al sistema de adquisición, sino que se precisa de accionamiento manual y extracción de las grabaciones de la misma forma.

Respecto a los sensores para el análisis de la mecánica vehicular, el cual es objeto de estudio del departamento de Ingeniería Mecánica y Fabricación de la ESI para comprender el impacto que ejercen las deformaciones de la vía sobre el vehículo ferroviario, se dividen en sensores IMUs y sensores láseres de medición de distancias mediante triangulación. A lo largo de este capítulo se profundiza en cada uno de estos sensores, mostrando sus características técnicas y conexionado.

2.1 Equipamiento sistema de adquisición

En esta primera sección, se muestra un esquema de los diferentes equipos que componen el sistema de adquisición de datos, mostrado en la figura 2-1. De este modo, el lector puede una visión general tanto de los diferentes equipos del sistema de adquisición como de las conexiones existentes entre ellos.

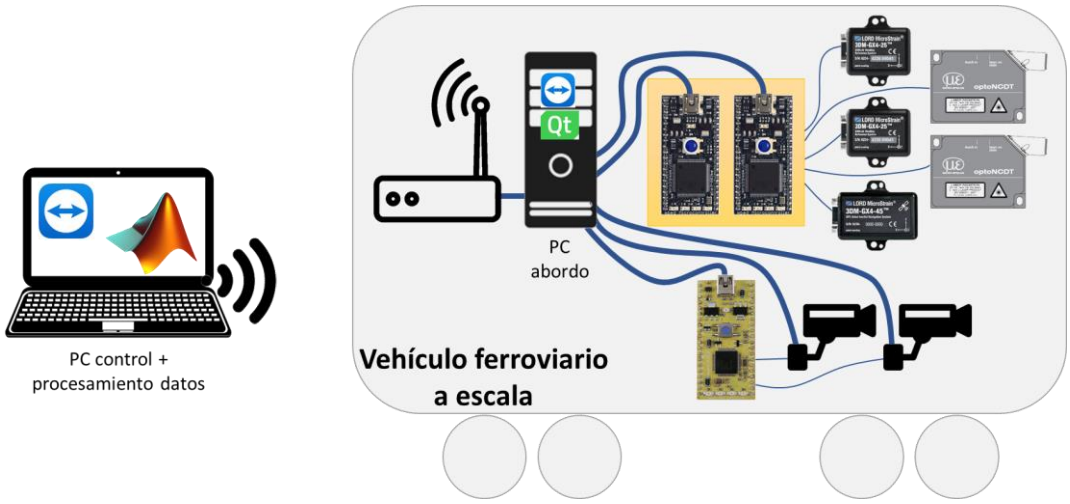


Figura 2-1. Esquema equipamiento y conexiones de sistema de adquisición de datos

Se puede dividir el sistema en dos grandes bloques, por un lado, se encuentra el PC de control y procesamiento de datos, que ejerce como torre de control del tren, y por otro lado los equipos localizados sobre el tren. El funcionamiento general de estos dispositivos se describe a continuación.

- PC control + procesamiento de datos. Este equipo es un ordenador portátil, ya que al ejercer como base de control del tren, al cual se conecta a su ordenador de abordo mediante WiFi, es necesario que pueda ser desplazado al lugar de los ensayos. Este ordenador preciosa de dos programas principales, TeamViewer y Matlab. El primero de ellos permite la conexión en remoto al escritorio del PC de a bordo del vehículo, al cual es necesario conectarse para ejecutar y parar el ensayo, así como para la adquisición de los ficheros de datos recogidos por los diferentes sensores que son procesados en el ordenador de abordo, a los cuales se les realiza el post-procesamiento.

Adicionalmente, durante el desarrollo de este proyecto, el cual puede considerarse parte del TO-BE y no del AS-IS, se ha incorporado la capacidad a dicho PC para el post-procesamiento de las señales adquiridas por los sensores, por lo que precisa de tener instalado el programa Matlab. Dicho post-procesamiento se describe en detalle en el capítulo 4 de este mismo documento.

- Router WiFi. Se sitúa sobre el vehículo ferroviario para capacitar la comunicación entre el PC de control y el PC de abordo. En necesario, previamente a los ensayos, habilitar una red WLAN para la comunicación entre ambos PCs, la cual es configurada manualmente haciendo uso de las IPs y MACs de ambos dispositivos.

- PC abordo. Este ordenador es el encargado de la transmisión de las señales de activación/parada de ensayos, así como de la recogida de paquetes de datos enviados por los microprocesadores y construcción de los ficheros de datos de salida. Este ordenador también precisa de TeamViewer para habilitar la conexión al mismo en remoto y poder ser controlado desde el PC de control. Adicionalmente, tiene instalado el programa Qt. Qt es un framework multiplataforma orientado a objetos, usado para desarrollar programas que utilicen interfaz gráfica de usuario [2]. Gracias a la interfaz gráfica desarrollada por el tutor de este mismo proyecto, Daniel García Vallejo, en dicha herramienta, este PC es el encargado del control del tren en los ensayos. Con un simple click en el botón Run en la interfaz de usuario, en la cual previamente se ha configurado los parámetros configurables del ensayo como es la velocidad a la que se desea que se desplace el tren. Adicionalmente, se envía una señal de activación para que los diferentes sensores, tanto cámaras de auscultación de la vía como sensores de estudio de la mecánica vehicular, empiecen a adquirir datos cuando se activa el movimiento del tren. Una vez se desea que finalice el ensayo, basta con presionar el botón Stop en la interfaz de usuario de Qt para detener la toma de datos por parte de los sensores y se pare el tren, haciendo una recopilación de los datos obtenidos en diferentes ficheros, uno por sensor, para su posterior estudio en el post-procesamiento de datos.

  A este ordenador se conectan las tres Mbed Microcontroller Board, que hacen de interfaz con los sensores de estudio de la mecánica vehicular, y las dos cámaras de auscultación de vías.

- Las tres Mbed Microcontroller Board, las cuales pueden considerarse en términos generales, ya que posteriormente se profundizará en ellas y su programación, como microprocesadores con diferentes puertos de entrada y salida a los que se conectan los sensores y son controlados por ellos. La dos azules son el LPC1768 y la amarilla el modelo LPC11U24. El primer modelo se encarga de la sincronización, activación y recogida de datos por parte de los sensores de estudio de la mecánica vehicular, mientras que el segundo modelo simplemente sincronización y activación de las cámaras de auscultación. Cabe decir que del primer modelo se han precisado dos placas porque una no proporcionaba toda la capacidad de pines necesarios para conectar los cinco sensores, así que se han conectado tres de ellos a una y dos a la otra.

- Los sensores mostrados se pueden dividir en dos grandes bloques: sensores para la auscultación de la vía, como son las cámaras, y sensores para el estudio de la mecánica vehicular, que son IMUs y sensores láser medidores de distancia mediante triangulación. El detalle de estos sensores es descrito en las siguientes secciones de este capítulo.

2.2 Sensores auscultación vía

Como es objeto de estudio la auscultación de la vía ferroviaria por parte del departamento Ingeniería Mecánica y Fabricación de la ESI, se procedió a la inclusión de sensores montados sobre el vehículo ferroviario para tal fin en su desplazamiento por las mismas. Cabe decir que dicha inclusión de sensores fue realizada por el personal del departamento en el trabajo previo a este comentado [1]. De este modo, no es necesario un amplio despliegue de sensores a lo largo de la vía, sino que el propio tren, a su paso por ellas, junto con los sensores y el post-procesamiento de estos datos, son los encargados de mostrar al analista una visión del estado de estas, ya que pueden deteriorarse y presentar imperfecciones con el desgaste del tiempo, o incluso elementos sobre las ellas como puede ser residuos del engrasado o pequeñas piedras.

Para ello, se han empleado dos sensores por cada vía para la auscultación de la misma, adicionalmente a uno que tendrá una visión general de la misma:

- Cámaras de vídeo de auscultación. Dos cámaras en el carbody enfocadas cada una hacia su carril correspondiente, derecho o izquierdo, que enfoca al láser de proyección lineal, grabando las deformaciones que presenta el mismo al ser proyectado sobre la vía y recorriendo la misma.
- Láseres de proyección lineal. Dos láseres en el carbody que se proyectan sobre la vía, cuya luz es grabada por las cámaras de auscultación.
- Cámara de vídeo GoPro. Esta tercera cámara de vídeo situada en la parte central del carbody graba la trayectoria del tren, así como los elementos de la vía.

En este capítulo se van a describir las características de los distintos sensores utilizados en el proyecto y se va a justificar su uso. Este proyecto es fundamentalmente experimental, es por ello que el número de sensores es mayor que el estrictamente necesario para la resolución del problema. La frecuencia de adquisición de los sensores está íntimamente ligada a la longitud de onda de las irregularidades. La frecuencia característica (en Hz) de las irregularidades depende de su longitud de onda característica y de la velocidad del vehículo ferroviario:

$$f = \frac{v}{\lambda} \tag{2–1}$$

donde v es la velocidad que puede alcanzar el vehículo en m/s y λ la longitud de onda característica de las irregularidades en metros. La frecuencia de adquisición de los sensores debe ser, al menos, diez veces la frecuencia característica para ser capaz de reconstruir las irregularidades de la vía [1]. En el caso de los ensayos realizados, el vehículo era capaz de alcanzar los 70 km/h, con una longitud de onda característica de 1 m se calcula una frecuencia característica de 20 Hz. Como la frecuencia de adquisición debe ser diez veces mayor que la frecuencia característica, se admitirá una frecuencia de adquisición mínima de 200 Hz para todos los sensores involucrados en la reconstrucción de las irregularidades.

2.2.1 Cámaras de video

La elección de las cámaras de video empleadas para el estudio de la auscultación de la vía y las características que presentan, se muestran en este apartado de forma resumida para que el lector tenga un mayor conocimiento de los mismos. Como la elección de las mismas no ha sido propósito de este proyecto, se hace referencia al trabajo involucrado realizado anteriormente para tal fin realizado en el departamento de Ingeniería Mecánica y Fabricación de la ESI de Juan Manuel Amador, tutorizado por José Luis Escalona. [1]

Para este cometido los sensores más adecuados son los sensores de distancia de barrido, los cuales devuelven la distancia hasta los distintos puntos en su ángulo de trabajo. Sin embargo, en su lugar, se utiliza una cámara de video y un láser de proyección lineal para la reconstrucción del perfil, ya que los sensores de distancia de barrido suponen un gran coste.

La cámara utilizada para esta función es la MQ003CG-CM de XIMEA. Se trata de una cámara CMOS a color con una resolución de 648x488 pixeles. Es una cámara muy pequeña (26x26x25 mm), con un peso de sólo 26 gramos y con la capacidad de hacer más de 500 FPS. La cámara se comunica mediante una conexión USB (3.0) [1].

Para sincronizar vía hardware las cámaras se requiere utilizar su conector de pines GPIO (General Purpose Input/Output). Este conector está formado por tres pines, uno de entrada uno de salida y tierra o común. El pin de salida puede ser utilizado para notificar acciones de la cámara a otro sistema electrónico. Sin embargo, se utiliza el pin de entrada para la sincronización. En este pin debe conectarse una señal digital cuyo valor alto corresponda a 24 V y el valor bajo a 0 v [1].

Conectando una señal cuadrada a la frecuencia a la que se quiere que la cámara haga una foto, ya que la cámara realizará una foto en cada flanco de subida o de bajada (pueden configurarse ambas opciones) de la señal. Si esta señal es común para las dos cámaras del vehículo, ambas realizarán la fotografía exactamente en el mismo instante y estarán totalmente sincronizadas.

Esta lente fue recomendada por el fabricante para el enfoque correcto de objetos entre 30 y 45 cms, distancia existente entre la cámara y el perfil de la cabeza del carril iluminado por el láser.

2.2.2 Láser de proyección lineal

El láser de proyección lineal tiene una potencia de 50 mW alimentado con una tensión de 5 V. Genera un haz plano láser a 120º desde su cabezal. El láser puede ser ajustado para conseguir una proyección nítida y de un espesor de aproximadamente 1 mm en objetos a distinta distancia, siendo el rango de varios metros [1].

2.2.3 Cámara de grabación del viaje

Al inicio del capítulo se enumeraron los sensores, entre ellos se encontraba una tercera cámara cuya función no es grabar la curva del perfil de los carriles.

Esta tercera cámara es una cámara ajena a la auscultación, que ha sido colocada para grabar toda la vía al completo, tanto los carriles como las traviesas, como otros elementos que puedan encontrarse en el recorrido del vehículo. Esta cámara es una Logitech HD Pro-Webcam C920, que presenta una resolución de 1920x1080 pixeles (Full HD), enfoque automático y el uso del formato de compresión H264 [1].

2.3 Sensores dinámica vehículo ferroviario

Adicionalmente a los nombrados, en el sistema de adquisición se incluyeron, gracias al trabajo de la referencia [1], los sensores para el estudio de la vía y del impacto que ejerce la misma sobre el tren, el cual puede sufrir desgaste si es expuesto durante un largo periodo de tiempo sobre una vía con imperfecciones.

Para tal fin, se han utilizado los siguientes sensores:

- Encoder. Es el encargado de convertir el movimiento en una señal eléctrica que puede ser leída por algún tipo de dispositivo de control en un sistema de control de movimiento, es decir, se utiliza para determinar la velocidad que presenta el motor del vehículo.
- IMU (Inertial Measurement Unit). Se han utilizado tres sensores inerciales en el vagón instrumentado. Uno de ellos situado en el centro de la cabina y los dos restantes en cada uno de los lados del bogie, justo en la parte superior de los amortiguadores. Además, se ha colocado un sensor inercial en cada uno de los vagones restantes (situados en la cabina) con el objetivo de conocer la influencia de la dinámica de un vagón en el resto del vehículo. Cada uno de estos sensores está orientado de forma que su eje x quede alineado con el eje longitudinal (dirección de avance del vehículo) [1].
- Láseres de triangulación. Situados en perpendicular en cada lateral del carbody midiendo la distancia entre este y el bogie.

2.3.1 Inertial Measurement Unit (IMU)

Las IMUs son las encargadas de proporcionar la mayoría de los datos necesarios para conocer la dinámica del vehículo, ya que estos sensores integran un acelerómetro, un giroscopio y un magnetómetro, todos ellos en los tres ejes (x,y,z).

2.3.1.1 Propiedades físicas, localización y conexionado dispositivos

Dos de las tres IMUs elegidas son el modelo 3DM-GX4-25. Una de ellas, la cual se sitúa en la parte trasera del carbody, puede verse en la figura de abajo.

Figura 2-2. IMU 3DM-GX4-25.

La tercera de ellas, la que va colocada en la parte central del bogie del vehículo es la 3DM-GX4-45, que tiene la misma funcionalidad que la 3DM-GX4-25 con el añadido de la función GPS. Sin embargo, esta funcionalidad no se utiliza. Como puede observarse en la siguiente figura, las características físicas de ambos modelos son muy similares.

Figura 2-3. Comparativa física modelos IMU 3DM-GX4-45 y 3DM-GX4-25. [3] [4]

El sensor dispone de dos puertos por los que comunicarse, ambos digitales. Una se trata de una salida USB y otro por el cual puede transmitir los datos por su puerto serie RS232. Por una parte, la conexión mediante USB exclusivamente se utiliza para la configuración del sensor, como es la velocidad de la transmisión de datos, que se muestra posteriormente. Por su parte, el puerto RS232 será mediante el cual el dispositivo se conecte al controlador, siendo así la entrada de la señal de activación, la salida de datos recogidos y la alimentación, como se puede ver en la siguiente figura:

sensor connector

sensor pin number	signal
1	USB D-
2	USB D+
3 **	+3.2V to +5.5 VDC
4	RS232 RxD
5	RS232 TxD
6 **	+5.2V to +36 VDC
7	PPS input
8	GND
9	--

** use either pin 3 OR pin 6 for sensor power

Figura 2-4. Pinout IMU 3DM-GX4-25 [1]

Esta figura tiene gran relevancia porque no se dispone de conector RS232 para los tres sensores inerciales, es por ello que la conexión se realiza de forma manual para la IMU mostrada en la figura 2-5, como se puede ver en la siguiente fotografía. Como es motivo de este proyecto que el lector pueda reproducir el escenario, se muestra en la figura 2-6 las conexiones de forma esquemática y detallada.

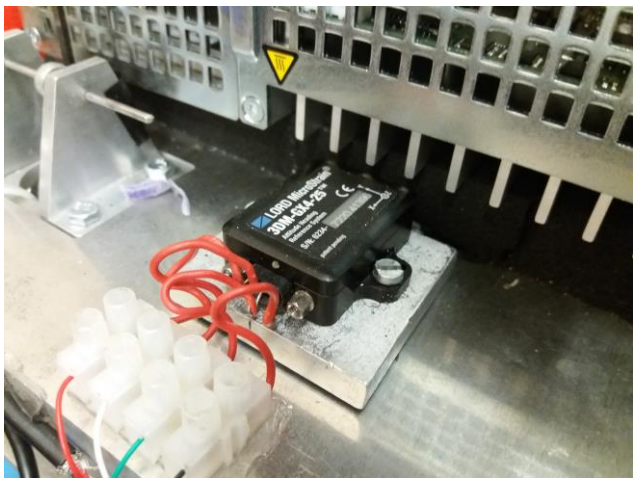
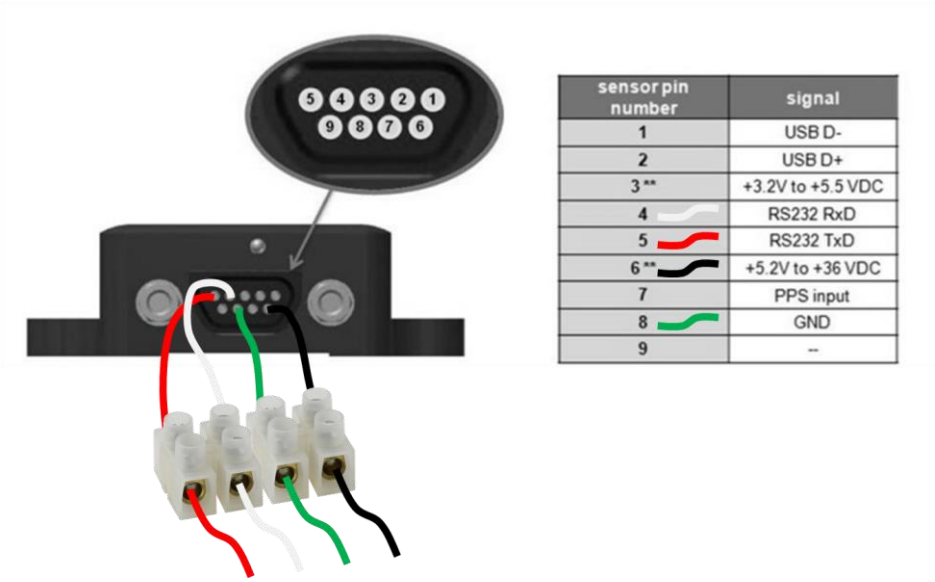


Figura 2-5. Conexionado IMU situada en carbody



sensor pin number	signal
1	USB D-
2	USB D+
3 **	+3.2V to +5.5 VDC
4	RS232 RxD
5	RS232 TxD
6 **	+5.2V to +36 VDC
7	PPS input
8	GND
9	--

Figura 2-6. Detalle conexionado IMU situada en carbody

Como aclaración de la toma de dicho pin de alimentación, el pin 3 de 3,2 V - 5,5 V es utilizado cuando se transmiten datos por el puerto USB y el pin de 5,2 V - 36 V con el puerto serie RS232 [1].

2.3.1.2 Configuración dispositivos

A continuación, se muestran las características que presentan cada uno de estos sensores [4]:

- El acelerómetro tiene un rango máximo de ±16g y una resolución de < 0,1mg con un ancho de banda máximo de 225 Hz.
- El giroscopio tiene un rango máximo de ±900º/s y una resolución de <0,008º/s con un ancho de banda máximo de 250 Hz.
- El magnetómetro no va a ser utilizado en este proyecto.
- Tanto el acelerómetro como el giroscopio tiene una máxima frecuencia de envío de paquetes de datos de 1000 Hz.

Las características configurables del sensor pueden ser modificadas mediante el software MIP Monitor del fabricante, como se ha realizado con la velocidad de conexión del puerto serie UART, clicando en la pestaña *Settings* y a continuación en *System*, cuya interfaz y desplegable puede verse en la siguiente figura.

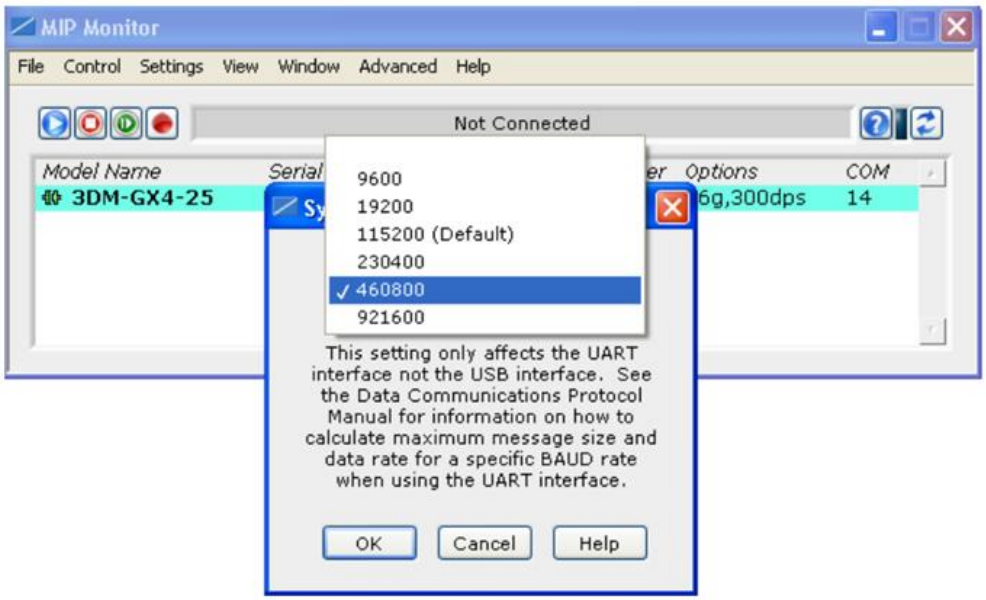


Figura 2-7. Configuración velocidad puerto UART de IMU [1]

2.3.1.3 Transmisión de datos dispositivos

Los tres sensores inerciales realizan la transmisión de datos en paquetes de bytes, los cuales presentan la estructura de la siguiente tabla:

Header					Packet Payload (2 fields)					Checksum	
SYNC1	SYNC2	Descript or Set	Payload Length	Field1 Length	Field1 Descript	Field1 Data	Field2 Length	Field2 Descript	Field2 Data	MSB	LSB
0x75	0x65	0x80	0x1C	0x0e	0x05	0x3E 7A 63 A0 0xBB 8E 3B 29 0x7F E5 BF 7F	0x0E	0x06	0x3E 7A 63 A0 0xBB 8E 3B 29 0x7F E5 BF 7F	0xB1	0x1E

Figura 2-8. Ejemplo paquete datos 3DM-GX4-25 con dos campos de datos

Se describen las diferentes partes del paquete emitido por las IMUs que se pueden observar en la tabla anterior:

- En la cabecera del paquete (Header) primero se encuentran dos bytes invariantes, 75 y 65 en hexadecimal, que equivalen a los caracteres 'u' y 'e' en el código ASCII. Estos bytes son útiles a para la programación de un algoritmo que detecte la llegada de un nuevo paquete.
- El tercer byte es un descriptor del paquete, que en el ejemplo es 0x80, que en este caso indica que el paquete es un paquete de datos AHRS (Attitude and Heading Reference System). [1]
- El cuarto byte de la cabecera indica el número de bytes de la siguiente parte del paquete Packet Payload, que en el ejemplo tiene una longitud de 14 bytes. [1]
- La siguiente parte del paquete es Packet Payload, que a su vez puede estar dividido en varios campos. Cada campo contiene un dato del sensor (un campo podría contener las aceleraciones y otro las velocidades angulares). Como se puede observar, cada campo tiene un primer byte que indica el tamaño del propio campo y un segundo byte que describe el contenido de los datos del campo. Tras ello, se encuentran un número variable de bytes que son los propios datos recogidos por el sensor. En este caso el número de bytes de estos paquetes es de 34 y los bytes contenedores de los datos son:
    - Aceleración eje x: Bytes 6 – 9
    - Aceleración eje y: Bytes 10 - 13
    - Aceleración eje z: Bytes 14 - 17
    - Velocidad angular respecto al eje x: Bytes 20 - 23
    - Velocidad angular respecto al eje x: Bytes 24 - 27
    - Velocidad angular respecto al eje x: Bytes 28 - 31

  Como se puede observar cada valor de aceleración o velocidad angular viene dado por cuatro bytes. Estos bytes corresponden a un número real según el estándar IEEE-754. El estándar IEEE-754 es el más extendido para representación de números en coma flotante. [1]

- La última parte del paquete Checksum son dos bytes que permiten comprobar si el paquete ha sido transmitido correctamente. Estos bytes son obtenidos al hacer operaciones binarias con los bytes del propio paquete. El sistema receptor puede hacer las mismas operaciones sobre los bytes del paquete recibido, en el caso de obtener los mismos valores del Checksum el paquete ha recibido contiene los mismos bytes que el enviado por lo que la transmisión ha sido correcta. [1]

Ahora se procede a calcular la velocidad a la que ha de ser configurado el sensor para la transmisión de los datos anteriores, teniendo en cuenta que los paquetes son emitidos a una frecuencia de 1KHz. Al ser una comunicación serie RS232, para cada byte, además de los 8 bit del propio byte, se envía un bit de inicio y uno de parada, que son en total 10 bit por byte enviado. Por lo que la mínima velocidad del puerto serie:

$$v_{min} = 10 \cdot N_{bytes} \cdot f \tag{2–2}$$

donde v_{min} es la velocidad mínima de configuración del puerto serie de la IMU para poder enviar todos los datos sin pérdidas, N_{bytes} el número de bytes del paquete de datos y f la frecuencia de envío de paquetes.

En el caso del paquete de datos de este proyecto, tiene una longitud de 34 bytes emitidos a una frecuencia de 1 KHZ, resulta un valor de 340000 baudios (bits por segundo) [1].

El sensor está configurado de fábrica para comunicarse por su puerto serie a una velocidad de 115200 baudios, por tanto, se aumenta la velocidad de comunicación del puerto serie de la IMU (y la velocidad del puerto del dispositivo electrónico receptor). El puerto serie tiene unas velocidades concretas a las que se puede configurar: [9600, 19200, 115200, 230400, 460800, 921600]. De todas ellas se ha elegido 460800 baudios por ser la más cercana por encima al valor calculado ya que, a mayor velocidad, mayor probabilidad de errores o de incompatibilidades cuando las velocidades son altas. La ejecución de la configuración puede consultarse en la sección *2.3.1.2 Configuración de dispositivos* de este mismo documento.

2.3.2 Láseres de triangulación

Los láseres de triangulación son sensores medidores de distancia que permiten conocer la distancia entre el carbody y el bogie en cada uno de los laterales del vehículo. Las mediciones de estas medidas serán micrométricas, es por ello que se precisa de sensores de gran precisión de medición.

2.3.2.1 Propiedades físicas, localización y conexionado dispositivos

El sensor de triangulación óptica optoNCDT 1302 es el modelo implementado en la red de sensores. El optoNCDT 1302 es un nuevo sensor de triangulación láser con tamaño compacto para mediciones de desplazamiento, distancia y posición. Debido a su tamaño extremadamente compacto que incluye un controlador integrado, el sensor también se puede integrar en un espacio de instalación restringido. Debido a su bajo peso, el optoNCDT 1302 es ideal para aplicaciones donde ocurren altas aceleraciones. El optoNCDT 1302 ofrece alta precisión y tasas de medición ajustables de hasta 2kHz. La compensación automática del objetivo proporciona un control de señal de distancia estable independientemente del color o brillo del objetivo. [5]

Figura 2-9. Sensor láser de distancia optoNCDT 1302

Este sensor proyecta un punto de luz sobre la superficie objetivo el cual es reflejado hacia un array de sensores ópticos. La posición del sensor óptico del array permite calcular la distancia relativa entre la superficie objetivo y el sensor como puede observarse en la figura.

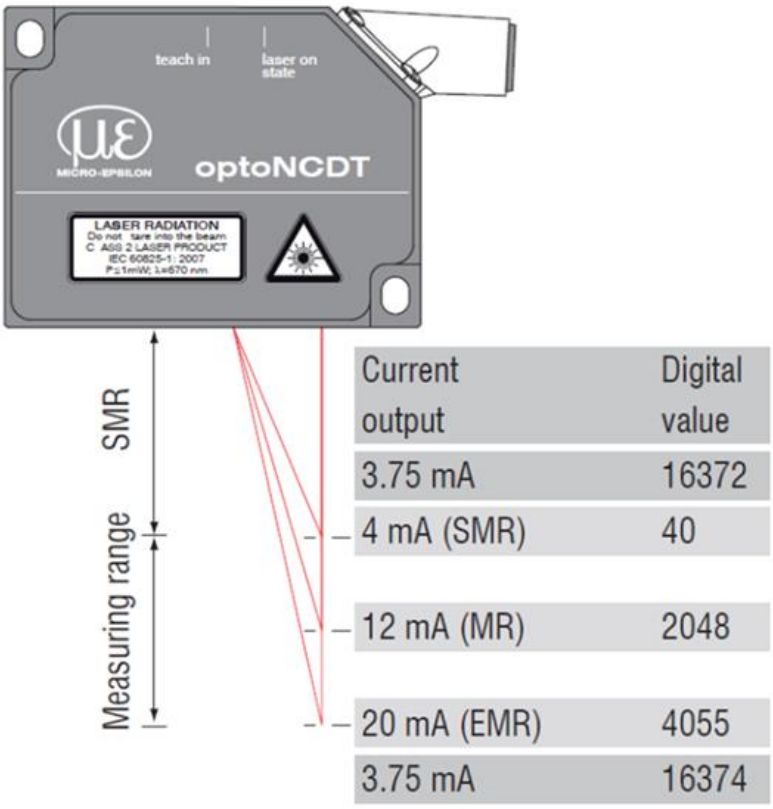


Figura 2-10. Salida sensor láser de distancia optoNCDT 1302. [6]

El sensor tiene un rango de 200 mm, con una precisión de 100 µm y una frecuencia de medición y transmisión de las medidas de 750 Hz. [6] Como puede observarse en la imagen 2-10, el rango de 200 mm no comienza en el cabezal del sensor sino a una distancia SMR (Start of Measuring Range) desde este. [6]

El sensor proporciona dos vías hardware por las que transmitir la información de las medidas. Una de ellas es una salida analógica cuya corriente será proporcional a la distancia medida. La corriente se moverá en el rango 4 - 20 mA siempre que la distancia medida esté dentro del rango del sensor. Cuando la superficie objetivo no está dentro del rango del sensor la corriente se fijará en 3,75 mA. Para utilizar está vía de información debe conectarse el pin, no olvidar conectar el pin GND. Además, el sensor debe tener conectado a GND el pin 8 para activarse. Por otro lado, existe un puerto digital por el que transmitir la información. Se trata de una salida serie RS422, pines 3-6 [1]. La definición de funcionalidad de los pines puede verse en la siguiente tabla. Una vez conectado a una fuente de alimentación de 24 V el sensor comienza a hacer medidas y enviarlas por su puerto serie.

Pin	Description		Color code PC1402-x/I	Specification
3	RS422 Rx+	Serial input	green	Internally terminated with 120 Ohm
4	RS422 Rx-		yellow	
5	RS422 Tx+	Serial output	grey	Terminate externally with 120 Ohm
6	RS422 Tx-		pink	
7	$+U_B$		red	11 ... 30 VDC, typical 24 VDC / 50 mA
8	Laser off	Switch input	black	Laser is active, if pin 8 is connected with GND
9	Teach in		violet	Connected to GND for at least 30 ms
10	Error	Switch output	brown	Open-Collector (NPN), I_{max} = 100 mA, U_{max} = 30 VDC, short circuit proof, turn off the power supply to reset the short circuit protection
11	I_{OUT}	4 ... 20 mA	white	R_{Load} = 250 Ω results U_{OUT} 1 ... 5 V with U_B > 11 V R_{Load} = 500 Ω results U_{OUT} 1 ... 10 V with U_B > 17 V
12	GND		blue	Supply and signal ground
1/2	n.c.			

Figura 2-11. Pinout optoNCDT 1302 [1]

2.3.2.2 Transmisión de datos dispositivos

Existen distintas posibilidades de leer los datos de este sensor. Una es conectarlo directamente a otro dispositivo electrónico con un puerto serie compatible, como un microcontrolador o un PC con este tipo de puerto. Sin embargo, también existe la posibilidad de usar un cable adaptador cuya salida es USB y así poder conectarlo a cualquier PC. Se emplea la primera opción para sincronizar todos los sensores mediante microcontroladores, los cuales se explicarán en capítulos posteriores [1].

Por cada byte enviado el sensor envía un bit de inicio y otro de final del byte y, entre ambos, los 8 bits del byte. En el primer byte se envía un bit a 1, el cual identifica qué byte se está recibiendo, seguido de los 7 bits más significativos del valor medido. En el segundo byte se envía un 0 seguido de los 7 bits menos significativos del valor de la distancia medida, como se puede ver en la siguiente tabla creada para una mejor comprensión del lector.

Start	1	7 Bit MSB							Stop	Start	0	7 Bit LSB								Stop
		1	D13	D12	D11	D10	D9	D8	D7			0	D6	D5	D4	D3	D2	D1	D0	

Figura 2-12. Paquete de datos optoNCDT 1302 con dos campos de datos

El resultado de los bits enviados no es la distancia en mm buscada, sino que es un valor comprendido entre los valores 40-4055 proporcional a la distancia. Si la distancia medida es menor que la mínima necesaria la que hay que localizar el sensor láser del objeto a medir, el valor enviado por el sensor será 16372. En caso contrario, si se sobrepasa el rango, se envía el valor 16374 como puede verse en la 2-10.

Para obtener el valor deseado en mm se debe hacer el siguiente cálculo:

$$distancia\ (mm) = \frac{rango\ (mm)}{rango_{max} - rango_{min}} \cdot V_r \tag{2–3}$$

donde V_r es el valor recibido por el sensor [1]. Siendo el rango 200mm, el $rango_{max}$ 4055 y el $rango_{min}$ 40, se obtiene:

$$distancia\ (mm) = 0{,}0498 \cdot V_r \tag{2–4}$$

El sensor necesita cuatro ciclos de 1,3 ms para calcular la medida y transmitirla después del ciclo de exposición en el que se recibe la onda reflejada. Esto significa que el dato recibido se refiere a la distancia leída por el sensor 4x1,3ms = 5,2ms atrás en el tiempo. [1]

3 ANÁLISIS SISTEMA FUNCIONAL AS-IS

Una vez conocida la arquitectura del vehículo a nivel general de todos los dispositivos y entrando en detalle en la descripción de los sensores, se realiza un análisis funcional del sistema intermedio de adquisición de datos incorporado en el tren a escala. Dicho sistema intermedio fue generado en el proyecto precedente a este ya comentado, *Instrumentación de un vehículo ferroviario para la auscultación dinámica con sistemas de visión artificial,* realizado por Juan Manuel Amador y tutorizado por José Luis Escalona.

Para comenzar, se muestra un esquema general del funcionamiento y flujo de datos entre los diferentes dispositivos del sistema de comunicación, para una mejor compresión del sistema por parte del lector. Una vez mostrada esta visión general, se realiza un estudio en profundidad del procesamiento de datos que se lleva a cabo en los dispositivos Mbed, ya que suponen el pilar fundamental en la gestión de los sensores. Como se ha comentado en capítulos anteriores, los sensores obtienen la información del mundo real, tanto de la dinámica vehicular como del sistema de auscultación de vía, y esta es enviada al PC de abordo una vez ha sido procesada por los microprocesador de las placas Mbed. Es por ello que el principal objetivo de este capítulo es mostrar al lector la funcionalidad de este sistema intermedio, así como el análisis del conexionado existente con los sensores para que pueda ser reproducible.

3.1 Funcionalidad del sistema de adquisición de datos

Para comenzar con esta sección, se enumeran las características que cumple el sistema de adquisición de datos de los sensores que componen la red, descritos en el documento [1]:

- Inicialización y control de los sensores. Es necesario enviarle a algunos sensores una señal en binario de inicio para que comience a transmitir datos o incluso mantener una señal cuadrada eléctrica en algunas de las entradas del sensor para que este continúe transmitiendo datos.
- Control de flujo de datos. Cada sensor utiliza un protocolo de comunicación distinto para transmitir los datos. Los paquetes de datos enviados por cada sensor son de distinto tamaño y la reconstrucción de los datos se hace de forma de distinta. Además, la frecuencia de envío de paquetes de cada sensor es diferente. El sistema es capaz de gestionar todos estos procesos sin pérdida de datos.
- Sincronización de sensores. El sistema procesa la información proveniente de los sensores para obtener los resultados, para unos resultados precisos el sistema debe tener un control riguroso del tiempo. Los sensores son a su vez un sistema independiente con su propio sistema de temporización, es por ello que el sistema de procesado envía cada uno de los paquetes de datos con una marca de tiempo respecto a una única referencia temporal, la que será utilizada para procesar los datos.
- Fiabilidad. El ruido corrompe los paquetes de datos produciendo resultados incorrectos. Las malas conexiones también pueden ser el origen de paquetes de datos erróneos o discontinuidades en la frecuencia de envío de datos.

A continuación, se presenta el esquema del flujo de datos, mostrando de forma general los pasos a seguir en la realización de un ensayo, el cual se verá en detalle en el capítulo 6 de este mismo documento. Con esta muestra se puede observar implícitamente que se cumplen los requerimientos expuestos anteriormente que ha de cumplir el sistema de adquisición de datos.

Para tal representación, se ha tomado como referencia el esquema creado para representar la figura 2-1, añadiendo flechas para indicar el sentido de la comunicación, diferenciadas por colores y formas según tipo de transmisión, y siendo enumeradas para su posterior explicación.

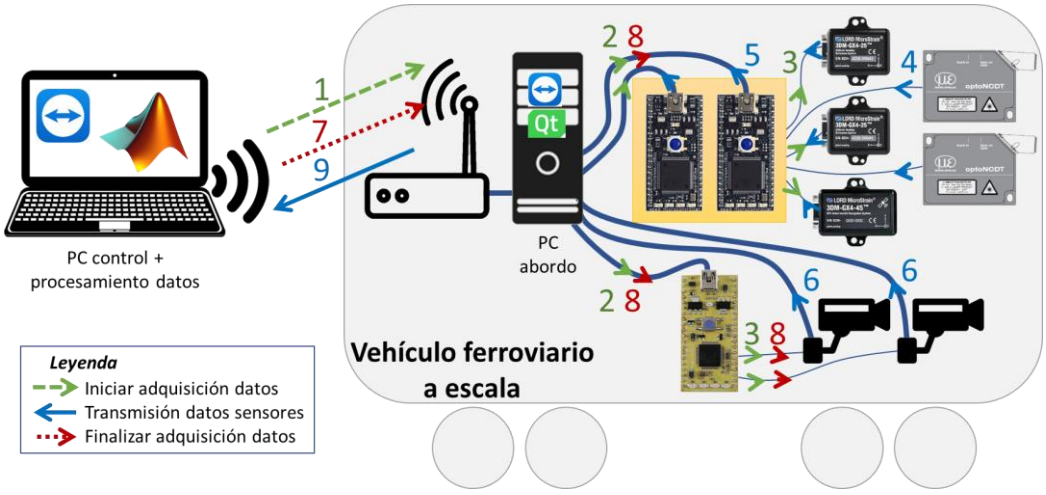


Figura 3-1. Esquema flujo de señales en sistema de adquisición de datos

A continuación, se describe la funcionalidad de las flechas enumeradas en el esquema anterior:

1. Inicialización de conexión desde PC de control a PC de abordo mediante señal WiFi, abriendo dicho acceso mediante la aplicación TeamViewer.
2. El PC de abordo transmite la señal de inicialización del ensayo a las placas Mbed, lo que supone activar los sensores de adquisición.
3. Las placas Mbed transmiten la señal de inicialización de adquisición de datos a los diferentes sensores que lo precisan, como son las IMUs y las cámaras de auscultación.
4. Los sensores de estudio de la dinámica vehicular comienzan a transmitir los datos obtenidos del mundo real a las placas Mbed modelo LPC1768.
5. Dichos dispositivos LPC1768, envían los paquetes de datos procedentes de los sensores con la estructura deseada para que el programa Qt del ordenador de abordo los reciba y lea correctamente.
6. Por su parte, las cámaras envían los ficheros de los vídeos que se encuentran en grabación directamente al PC de abordo.
7. Mediante la conexión remota, lo cual hay que entender que realmente se ejecuta en el monitor del PC de abordo y no en el PC de control, se realiza la acción de parada de adquisición de datos en la interfaz de usuario en Qt desde el PC de control.
8. Esta señal de parada es transmitida a las placas Mbed para que dejen de recibir datos de los sensores, así como se envía la señal de parada de grabación a las cámaras de auscultación.
9. Finalmente, una vez se ha parado el ensayo, se realiza el volcado de los ficheros de los datos obtenidos mediante el sistema de adquisición al PC de control mediante la conexión remota para poder estos ser post-procesados en Matlab.

3.1.1 Sistema intermedio cámaras de vídeo y encoder

Como se ha visto en el flujo de datos anterior, el dispositivo principal de este sistema intermedio es el microcontrolador Mbed LPC11U24 de NXP. ARM Mbed es una plataforma de prototipado rápido que facilita el montaje y programación de distintos microcontroladores. La plataforma Mbed ofrece un compilador online con el que programar todos los microcontroladores compatibles, sin necesidad de hacer grandes cambios para migrar el código de un microcontrolador a otro. El lenguaje utilizado en la programación es C++.

La placa de prototipo utilizada es la que integra el microcontrolador LPC11U24. Se trata de un microcontrolador de 32 bit y 48 MHz programable por un puerto USB. El microcontrolador se integra en una pequeña placa en formato PDIP con los circuitos de alimentación, conexión con el PC, memoria externa para guardar archivos binarios y otras funciones ya diseñados e integrados. Como se puede ver en la figura 3.2, entre la salidas de la placa se dispone de un puerto serie, un puerto I2C, un puerto USB, dos puertos SPI, ocho canales de entrada a un conversor ADC y una gran cantidad de puertos GPIO. El microcontrolador puede ser alimentado directamente mediante el USB de programación y comunicación con el PC. Tiene dos pines para alimentar periféricos de 3,3 V y 5 V [1].

La conexión entre la placa Mbed y el encoder se ha realizado conectando el pin de alimentación del sensor al pin VOUT (ver figura 3-2) de la placa Mbed, y uniendo las tierras ya queda alimentado el sensor. El pin de señal del sensor se puede conectar a cualquier pin GPIO de la placa. En este caso se ha conectado un sensor en el pin 19 y el otro en el pin 20.

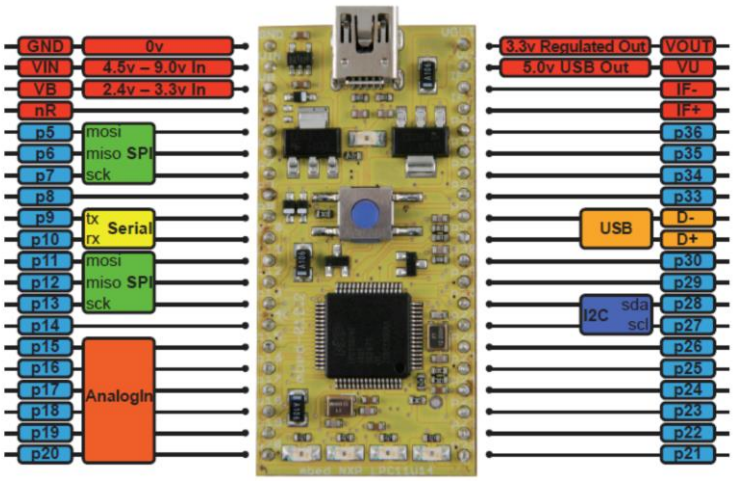


Figura 3-2. Pinout Mbed LPC11U24

Como es objetivo de este proyecto mostrar los detalles para que el sistema de adquisición sea reproducible, se adjunta en el Anexo 1 la programación en C++ que incorpora el microcontrolador que permite leer el encoder y enviar la señal de sincronización a las cámaras. Dicho código ha sido proporcionado por el personal del departamento de Ingeniería Mecánica y Fabricación de la ESI, siendo objeto de desarrollo en el proyecto previo a este [1], es por ello que es de su propiedad intelectual ya que no forma parte del desarrollo de este proyecto.

3.1.2 Sistema intermedio sensores dinámica vehicular

Las ventajas que presenta el empleo de este sistema intermedio respecto a la conexión directa de los diferentes sensores al PC de abordo son las siguientes [1]:

- La principal ventaja es el control sobre el proceso de adquisición que se consigue mediante electrónica especializada para realizar esta tarea. La primera propuesta utiliza un PC con un sistema operativo (Windows) que es el que controla todo el hardware y en numerables ocasiones el acceso a un nivel bajo de control de hardware es complejo o no está permitido. Es por ello que en los ensayos realizados se producen picos en los periodos de adquisición. Además, a pesar de la gran potencia de procesado de un PC comparado con un microcontrolador común es muy alta, este tiene que atender muchas otras tareas, como el control de periféricos o los recursos que consume el propio sistema operativo. Existen dos formas de solucionar este problema, una es usar un sistema operativo de tiempo real que asegure los tiempos de lectura y escritura con una precisión aceptable y otra es diseñar una electrónica especializada como se ha propuesto en este proyecto. Los microcontroladores permiten el acceso a todos sus procesos a nivel hardware y un control muy preciso del tiempo que va a tardar en realizarse cada tarea y el orden en el que se realizan.
- El PC queda libre de la tarea de adquisición y puede dedicar sus recursos al procesado de los datos.
- Sincronización. La sincronización con un microcontrolador es muy sencilla, sólo se tiene que acompañar cada dato con una marca de tiempo, leyendo el reloj del microcontrolador en el instante en el que se recibe el dato.

- Comunicación directa con los sensores. En la propuesta los sensores son leídos utilizando las librerías de funciones que proporciona la API de cada sensor y los driver instalados. Aunque estas funciones suelen estar muy optimizadas, muchas veces no están completamente documentadas, por lo que no es posible conocer cómo se ejecuta cada función y cuánto tiempo tarda en hacerlo. La abstracción que ofrecen las funciones puede ser una desventaja en proyectos de gran precisión temporal y sincronización como el presente proyecto. La comunicación con el sensor utilizando microcontroladores obliga al aprendizaje de los comandos del sensor y sus protocolos de comunicación, pero a su vez asegura el control de todas las posibilidades del sensor y su sincronización con otros sensores.

El microcontrolador utilizado en este sistema intermedio es el Mbed LPC1768 de NXP [1]. Este microcontrolador pertenece a la plataforma Mbed. Se trata de una versión con mejores características que el microcontrolador utilizado en el sistema de sincronización de cámaras, también se puede programar con la API online de la plataforma Mbed. Este microcontrolador de 32 bit con una frecuencia de 96 MHz tiene tres puertos serie, dos puertos I2C, un puerto USB, tres puertos SPI, ocho canales de entrada a un conversor ADC, ocho salidas PWM (Pulse Wrigth Modulación), una salida analógica y una gran cantidad de pines GPIO. El pinout de la plataforma puede encontrarse en la siguiente figura:

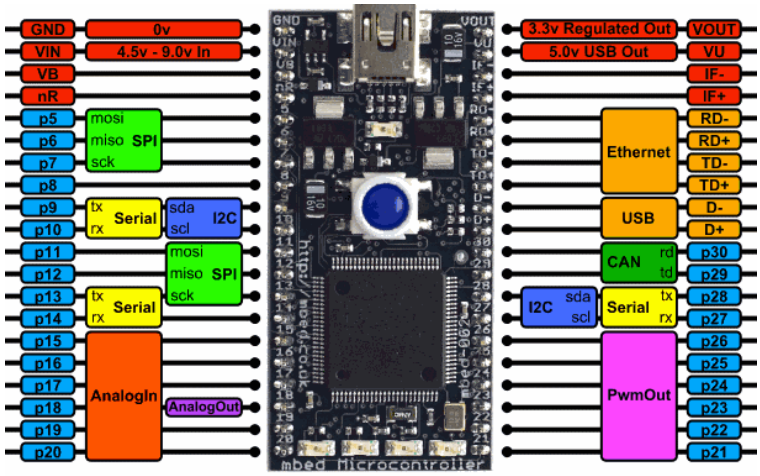


Figura 3-3. Pinout Mbed LPC1768

El sistema electrónico diseñado que controla el flujo de datos está compuesto por dos plataformas Mbed con el microcontrolador LPC1768. Cada microcontrolador Mbed LPC1768 tiene tres puertos serie, insuficientes para leer los tres sensores inerciales y los sensores de distancia, es por esto que se han utilizado dos microcontroladores, que suman en total seis puertos serie. [1] Por razones de simetría y de distinto tamaño de flujo de datos (los sensores inerciales tienen un flujo de datos mucho mayor que los sensores de distancia) se han distribuido los sensores de la siguiente forma:

- Mbed 1: Esta Mbed controla el flujo de datos de el sensor inercial y el sensor de distancia de uno de los lados del vehículo. También controla el sensor inercial central situado en el bogie. El sensor de distancia va conectado al puerto serie de los pines 27 y 28 y el sensor inercial en el puerto serie de los pines 9 y 10. El sensor inercial colocado en la parte central del bogie es conectado al puerto serie de los pines 13 y 14.

- Mbed 2: Esta Mbed controla el flujo de datos de el sensor inercial y el sensor de distancia del otro lado del vehículo. El sensor de distancia va conectado al puerto serie de los pines 27 y 28 y el sensor inercial en el puerto serie de los pines 9 y 10.

Cada Mbed envía los datos de los sensores conectados a él hacia el PC, es por esto por lo que existen dos conexiones USB entre el sistema microcontrolador y el PC. Por estas mismas conexiones USB se alimentan las plataformas Mbed. Se puede ver el aspecto de la placa con los microcontroladores y los conectores para los sensores en la siguiente figura.

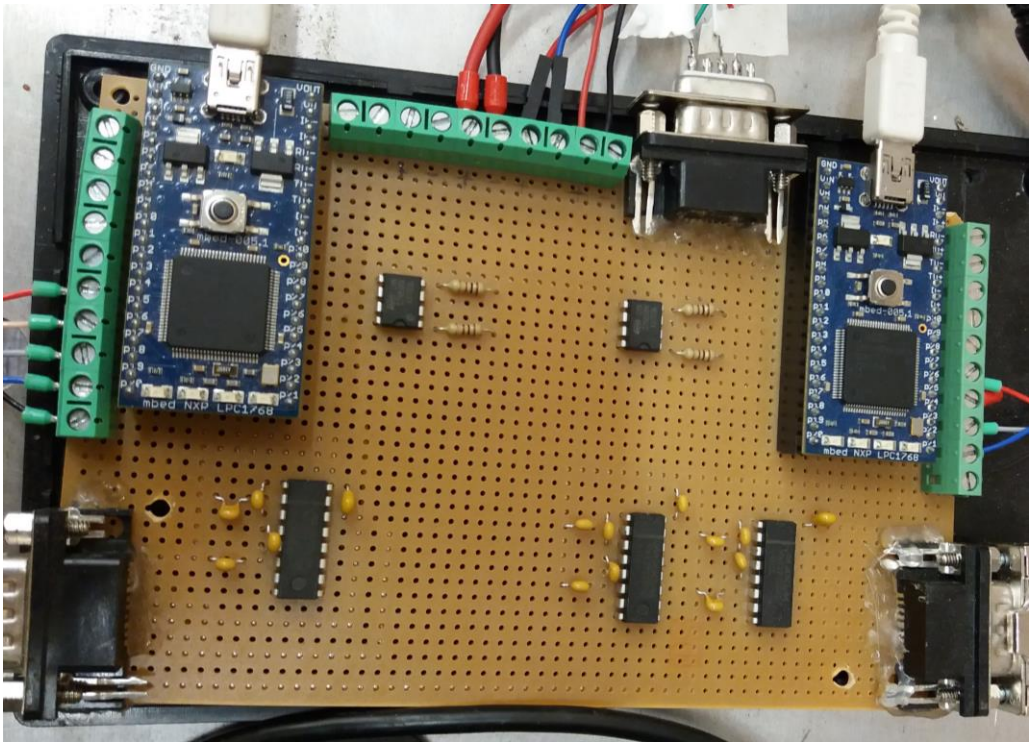


Figura 3-4. Sistema intermedio sensores dinámica vehicular.

Un problema que puede aparecer por el uso de dos microcontroladores distintos, cada uno con su propio temporizador, es una mala sincronización entre los sensores leídos por un microcontrolador y el otro. Este problema no debería aparecer ya que cuando el PC manda los comandos de inicio de adquisición a los microcontroladores, el tiempo entre que llega el mensaje al primer microcontrolador y luego al segundo, es de decenas de microsegundos, un tiempo despreciable en la escala de tiempo de ejecución del ensayo.

3.1.3 Conexionado placa sistema intermedio

La conexión entre los sensores inerciales y de distancia, y los microcontroladores, no es inmediata debido a los distintos niveles de tensión utilizados por los sensores y los puertos serie UART de los microcontroladores. Es por ello que en esta sección se estudia el conexionado que presenta el sistema intermedio de sensores para estudio de dinámica vehicular, la cual se denomina para abreviar "placa sistema intermedio". El conexionado que presenta puede verse en la siguiente imagen 3-5. Para un estudio más exhaustivo, se ha creado el documento adjunto en el Anexo 4 de este documento, donde se presentan todas las conexiones existentes desde las Mbed Boards hasta cada uno de los conectores de cada sensor.

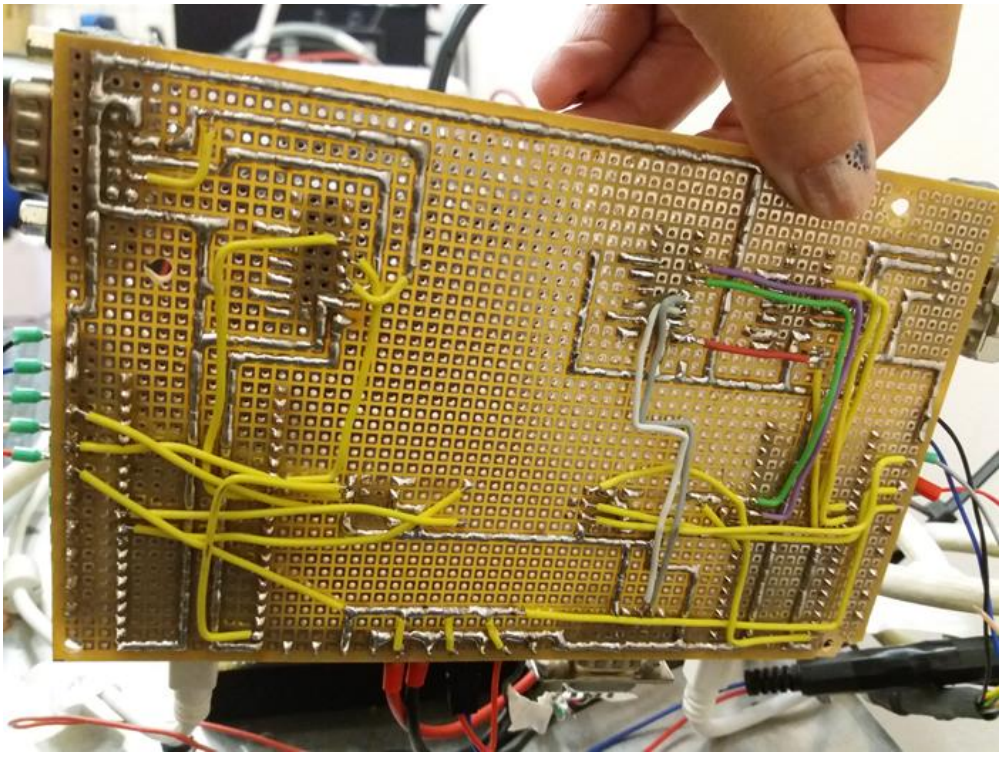


Figura 3-5. Conexionado sistema intermedio sensores dinámica vehicular.

3.1.3.1 Conexión Láser medidor distancia – Mbed Board

Los niveles de tensión con los que se comunica el sensor, que sigue las especificaciones de RS422, no son compatibles con los niveles TTL de los microcontroladores. Para la conversión de los niveles de tensión se ha utilizado el integrado ST485 (también serviría el MAX485, MAX3485 o similares). [1] Como puede observarse en el documento del anexo 4, el puerto serie del sensor tiene cuatro pines (Ti+, Tx-, Rx+ y R.-), dos para la recepción y dos para la transmisión. Como no se va a enviar ningún dato al sensor solo se utilizarán los de transmisión como se puede ver en el esquema de la anexo 4. El integrado va alimentado con la salida de 5V del microcontrolador [1]. Las dos resistencias conectadas a los pines TX y RX sirven para limitar la corriente y proteger dichos pines.

3.1.3.2 Conexión IMU – Mbed Board

El puerto serie de este sensor también tiene diferentes niveles de tensión que la UART de la Mbed. En este caso el sensor sigue las especificaciones RS232 [1]. Para hacer la conversión a niveles TTL se ha usado el integrado MAX3232 [1].

Se van a utilizar los pines R y Ti del puerto serie RS232, además, debe conectarse la tierra y la alimentación. Llama la atención que el sensor tiene dos pines con los que se puede alimentar el sensor. El pin 3 es el pin por el que se alimenta cuando se usa la comunicación USB. Sin embargo, para comunicarse por el puerto serie RS232 debe alimentarse por el pin 6 con una tensión entre 5,2 V y 36 V [1].

En la siguiente figura se muestra el esquema de conexiones de estos pines con el integrado que hace la conversión de niveles de tensión y de este con el microcontrolador. Esta figura se ha obtenido del proyecto [1].

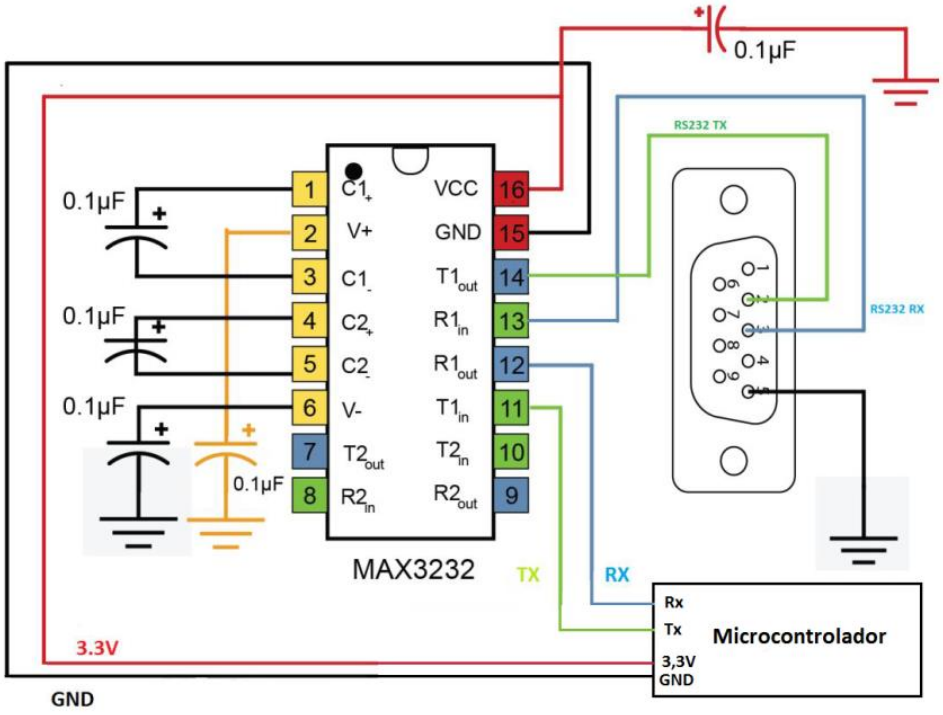


Figura 3-6. Esquema conexión Mbed Board e IMU [1].

3.1.4 Mbed Board para lectura de sensores

Como se ha comentado, es preciso que la frecuencia de adquisición por parte de la placa Mbed sea igual o mayor que la frecuencia con que los sensores le envían los datos a ésta. Por su parte, el sensor de distancia emite datos a 750 Hz y el sensor inercial a 1 KHz [1].

Al igual que con el sistema intermedio de las cámaras de auscultación, no objeto de este proyecto el desarrollo de la programación de estas LPC1768, ya que ha sido desarrollado por el proyecto llevado a cabo por Juan Manuel Amador y tutorizado por José Luis Escalona [1]. Sin embargo, en esta ocasión se ha realizado un estudio más exhaustivo del código contenido en dichos dispositivos para una mejor comprensión del tratamiento de los datos, que luego serán objeto de análisis de este proyecto, ya que las señales tanto de las IMUs como de los sensores medidores de distancia sí serán procesados por estas Mbed Board, lo cual no ocurría con las cámaras de video al estar conectadas directamente al ordenador y sólo reciben de la placa la señal de sincronización. El código se puede encontrar en el Anexo 1, el cual es propiedad del personal del departamento de Ingeniería Mecánica y Fabricación.

El programa de los microcontroladores se ha compilado en la API online de Mbed. El programa comienza con la declaración de las variables y la configuración de la velocidad de los puertos serie [1]:

- El puerto serie de comunicación con el sensor de distancia se configura a 115200 baudios, que es la velocidad por defecto del sensor.
- El puerto serie de comunicación con el sensor inercial se configura a 460800 baudios, como se comentó en la sección *2.3.1.3 Transmisión de datos dispositivos* de este documento. El puerto de comunicación con el PC debe ser capaz de enviar los datos de todos los sensores. Como el paquete de el sensor inercial que se envía al PC es de 30 bytes a una frecuencia de 1 KHz y 8 bytes del paquete con los datos del láser a 750 Hz, esto suma en total 30B·1000Hz + 8Bx750Hz = 36000 bytes por segundo que el microcontrolador envía al PC por el puerto serie. Como se calculó en el capítulo anterior, la configuración mínima de velocidad debe es de 360000 baudios, por ello, se ha configurado a 460800 baudios.

Una vez hechas las inicializaciones y configuraciones necesarias, el programa entra en un bucle infinito, en el que se realizan las siguientes acciones [1]:

- Se comprueba si han llegado los bytes procedentes del PC. De forma similar al tacómetro, un byte con el carácter 'a' indica el comienzo de la adquisición y la 's' el final de esta. Cuando se inicia la adquisición se activa una variable semáforo que permite la reconstrucción y envío al PC de los datos de los sensores, se reinicia el temporizador, se inicia y se enciende un led indicador. También se envía un paquete de bytes al sensor inercial que le indica que debe comenzar a enviar datos por streaming. Cuando finaliza la adquisición, se apaga el led indicador, se para el temporizador y se resetea.
- Se comprueba la llegada de bytes del sensor inercial y se realizan las operaciones necesarias. Sólo se hace si esto si no se está procesando un paquete del sensor de distancia.
- Se comprueba la llegada de bytes del sensor de distancia y se realizan las operaciones necesarias. Sólo se hace si esto si no se está procesando un paquete del sensor inercial.

Cuando llega el primer byte de un paquete de un sensor, se deja de comprobar la llegada de bytes de otros sensores. Esto se hace para que no se mezclen los distintos paquetes que se envían al PC con datos de distintos sensores, ya que cuando llega un byte de un sensor, a continuación, se procesa y se envían los bytes correspondientes al PC. El tiempo que tarda el microcontrolador en recibir todos los bytes del paquete enviado por un sensor es muy pequeño. Si durante este pequeño periodo de tiempo llegara un byte procedente de otro sensor, este simplemente se guarda en el buffer hardware del puerto serie del microcontrolador hasta que se vuelve a preguntar por la llegada de un dato a ese puerto, por lo que no existe pérdida de bytes [1].

3.1.5 Procesado del paquete del sensor de distancia

En el capítulo anterior se describió el formato del paquete del sensor de distancia y como debía ser reconstruido. Una vez leído el byte del sensor de distancia, lo primero que hace el programa es distinguir si es el byte más significativo o el menos significativo. Esto es sencillo gracias a que el byte más significativo siempre tiene el bit más significativo a 1 y el byte menos significativo siempre lo tiene a 0 [1]. Una vez se ha comprobado que el byte es el más significativo se guarda en una variable entera y se pone a 0 el bit más significativo, para que no influya en el valor reconstruido.

La variable byteIN guarda el byte recibido. Una vez hecho esto se activa una variable semáforo que permite terminar la reconstrucción con el byte menos significativo. Esto asegura que ya se ha procesado un byte más significativo (el primero que envía el sensor) antes de calcular la distancia. Cuando se recibe el byte menos significativo, se lee el tiempo. El tiempo que pasa entre la llegada del primer y del segundo byte es despreciable en la escala de tiempo del presente proyecto por lo que puede leerse el tiempo cuando llega el primer o segundo byte del sensor sin pérdida de precisión [1].

Antes de enviar el paquete con la información recogida hay que terminar la reconstrucción de la medida de la distancia. Para conseguirlo, los siete bit más significativos guardados anteriormente en la variable distancia se desplazan siete posiciones a la izquierda, y al resultado se le suma directamente el valor del byte recibido (es equivalente a hacer una operación OR entre bit de ambas variables en este caso), el byte menos significativo. Con esto se obtiene la reconstrucción. El valor reconstruido es un valor en unidades de ingeniería (valor devuelto por el ADC interno del sensor), para su conversión a unidades reales en mm debe ser multiplicado por 0,0498 como se vio en el capítulo anterior. [1]

Una vez reconstruido el valor de la distancia, se divide en dos bytes, primero se guarda el byte menos significativo, se hace un desplazamiento de los bit ocho posiciones a la derecha, y se guarda el byte más significativo. Todos los bytes que se van a enviar se guardan en un buffer dónde se guarda el paquete del sensor de distancia. Los dos primeros bytes de este paquete son dos caracteres 'L'. Esto es así para que el PC sea capaz de diferenciar este tipo de paquetes de los paquetes del sensor inercial [1]. Los últimos cuatro bytes del paquete son los bytes de la variable del tiempo leído anteriormente, quedando el paquete con los datos del sensor de distancia enviado al PC con el formato de la siguiente figura:

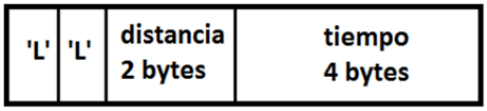


Figura 3-7. Formato paquete de datos sensor de distancia enviado al PC por Mbed Board [1].

3.1.6 Procesado del paquete de IMU

En el capítulo anterior se describió el formato del paquete del sensor inercial y los bytes que contienen los datos de aceleraciones y velocidades angulares. Debido a la complejidad del paquete enviado por el sensor inercial, se ha utilizado una variable que almacene el número de bytes recibidos por el sensor que pertenecen a un mismo paquete (variable nBytes). Una vez recibido un byte por el puerto serie conectado al sensor inercial, primero se comprueba que el byte recibido corresponde a los bytes de inicio de paquete y no se están recibiendo bytes erróneos o de datos [1].

Hay que recordar que los paquetes enviados por estos sensores comenzaban siempre con los bytes ’u’, ’e’, por lo que cada vez que se recibe un byte se comprueba si es una ’e’ y el anterior una ’u’ (cada vez que llega un nuevo byte se guarda el anterior). Además, para asegurarse que no son dos bytes de datos de un paquete que coincidan con estos caracteres, se comprueba que el número de bytes recibidos del ´último paquete es igual o mayor al número de bytes de los paquetes enviados por el sensor (variable nBytes). Es por esto por lo que se inicializa la variable nBytes con el tamaño del paquete del sensor, sino se hiciera esto nunca comenzaría la reconstrucción de ningún paquete, aunque sea recibido correctamente por el puerto serie. Una vez se reciben caracteres ’u’, ’e’, se envía por el puerto serie dos caracteres ’I’, se guarda el tiempo y se cuentan los bytes recibidos de ese paquete entrante. Cada vez que se recibe un byte de datos de aceleraciones o velocidades angulares simplemente se reenvía hacia el PC, el resto de bytes son ignorados [1].

Cuando se recibe el ultimo byte de datos, se reenvía y además se envía el tiempo guardado anteriormente dividido en 4 bytes, enviando primero el byte más significativo [1]. El paquete resultante enviado al PC tiene el formato mostrado en la siguiente figura:

'I'	'I'	aceleración X 4 bytes	aceleración Y 4 bytes	aceleración Z 4 bytes	vel. angular X 4 bytes	vel. angular Y 4 bytes	vel. angular Z 4 bytes	tiempo 4 bytes

Figura 3-8. Formato paquete de datos de IMU enviado al PC por Mbed Board [1].

4 POST-PROCESAMIENTO SEÑALES SENSORES

El tratamiento de los datos capturados por los sensores que son foco de estudio para conocer el estado de la vía y la dinámica del vehículo ferroviario, puede denominarse como el principal objetivo del conjunto de proyectos que se llevan a cabo, entorno el tren a escala, en el departamento de Mecánica y Estructura de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla.

El post-procesamiento de las señales supone la creación de algoritmos para el análisis de los datos obtenidos por los sensores y la representación de los mismos de forma gráfica para que el analista de dichos datos pueda estudiarlos y sacar conclusiones sin suponer mucho trabajo adicional por parte de esta persona. Es por ello que en este capítulo se trabajan los tres bloques siguientes, el primero de ellos engloba la representación de los datos, el segundo la automatización de dicho proceso de tratamiento de ficheros de datos y representación y, como tercer bloque, la mitigación de datos espúreos. Gracias a la agilización en el tratamiento de los datos tras realizar un ensayo, se ha incorporado la capacidad de visualizar los datos inmediatamente después de ser recogidos por los sensores en un ensayo, lo cual supone un gran avance ya que, anteriormente, este proceso suponía volver a las instalaciones de la universidad para verificar la validez de las pruebas realizadas, sin margen para poder realizar mejoras sobre el tren en el momento del ensayo o, incluso, movilizando de nuevo todo el proceso logístico para realizar unos nuevos ensayos al no ser válidos los anteriores.

4.1 Representación de datos de cada sensor

La representación de los datos de los diferentes sensores se ha realizado, como se ha comentado anteriormente, utilizando el programa Matlab, el cual es ampliamente conocido en la Escuela Técnica Superior de Ingeniería por sus capacidades para la implementación de algoritmos, la representación de datos y la comunicación con otros programas. [7]

Para poder realizar un análisis de las necesidades que presenta el programa Matlab para obtener como inputs los documentos de salida del tratamiento de la señal del procesado intermedio, se han de abrir de forma previa post-procesamiento análisis y en formato texto los diferentes ficheros de salida del procesamiento de datos y añadir el símbolo *%* a las primeras líneas de cada uno donde se encuentra el texto de cabecera. En el caso de que este símbolo ya esté incluido al principio de la cabecera de un fichero, no será necesario incluirlo de nuevo.

A continuación, se explican los diferentes módulos que se han creado en lenguaje m (propio de Matlab) para representar los datos procedentes de los diferentes sensores de estudio de la dinámica vehicular, como son el encoder, las IMUs y los láseres de triangulación, los cuales se encuentran recogidos dentro del fichero matlab *full_representation.m.* Este fichero nos permite representar los diferentes ficheros que contienen los datos recogidos por los sensores de forma individual, es decir, cada fichero ha de pasarse como parámetro de la función del mismo nombre del fichero, de lo contrario, si un fichero no se quiere representar o no contiene datos, se ha de poner un 0 en su lugar.

A continuación, se describen los diferentes parámetros de entrada que han de pasarse a la función:

- encoder_file: fichero de salida encoder del motor, *_enc.csv (donde * variará en cada fichero, siendo la marca de tiempo donde se ha realizado el ensayo), el cual consta de 6 columnas:
    - time: representa el triple de la diferencia entre dos muestras consecutivas de la columna "elapsed time (ms)" (NOTA: no es utilizado para las representaciones de este fichero).
    - omega_m (rpm): velocidad del motor en revoluciones por minuto (NOTA: si es negativo la polarización a la que se ha conectado el moto es correcta).
    - encoder_position: no es utilizado en la representación de los datos al no ser relevante para tal fin.
    - position_change: no es utilizado en la representación de los datos.
    - elapsed time (ms): instante de tiempo de toma de datos.
- imu1_file: fichero de salida IMU 1 conectada al puerto serie 1 (SP1, 2 IMUs conectadas a él), (*_sp1imu.csv), 7 columnas:
    - Time (us): instante de tiempo de toma de datos.
    - Angular rate X (º/s): velocidad angular obtenida entorno al eje x, obtenida en grados/segundo.
    - Angular rate Y (º/s): velocidad angular obtenida entorno al eje y, obtenida en grados/segundo.
    - Angular rate Z (º/s): velocidad angular obtenida entorno al eje z, obtenida en grados/segundo.
    - Acceleration X (g): aceleración obtenida entorno al eje x, obtenida en g.
    - Acceleration Y (g): aceleración obtenida entorno al eje y, obtenida en g.
    - Acceleration Z (g): aceleración obtenida entorno al eje z, obtenida en g.
- imu2_file: fichero de salida IMU 2 conectada al puerto serie 1 (SP1, 2 IMUs conectadas a él), (*_sp1imu2.csv). Mismas columnas que imu1_file.

- imu3_file: fichero de salida IMU 3 conectada al puerto serie 2 (SP2, 1 IMU conectada a él), (*_sp2imu.csv). Mismas columnas que imu1_file.
- laser1_file: fichero de salida laser 1 conectado al puerto serie 1 (SP1, 1 laser conectado a él), (*_sp1laser.csv), 2 columnas:
    - Time (us): instante de tiempo de toma de datos.
    - distance (mm): distancia desde la salida del laser hasta donde se proyecta el punto del mismo (NOTA: para correctas mediciones el led del laser ha de estar en verde).
- laser2_file: fichero de salida laser 2 conectado al puerto serie 2 (SP2, 1 laser conectado a él), (*_sp2laser.csv). Mismas columnas que laser1_file.
- time_file: muestra el tiempo de ejecución del ensayo. Siempre hay que ponerlo.

Todo el código expuesto queda a disposición del lector en el Anexo 2, al final de este mismo documento.

Cabe destacar que, para una mejor comprensión de todo aquel que lo precise y siguiendo las buenas prácticas de programación, se han incluido comentarios tanto a en las primeras líneas de cada fichero, definiendo la funcionalidad del mismo y los diferentes inputs y outputs de la función, como intercalado en el código para explicar el funcionamiento del mismo. De este modo, tal y como era requisito de este proyecto, cualquier otra persona que adopte o necesite utilizar el código expuesto, pueda reproducirlo y entenderlo sin gran dificultad, ya que, además, todo está tabulado y hecho de la forma más estructurada y simplificada posible.

Una vez explicados los argumentos de entrada de la función y la forma en que se ha programado, se expone el contenido de los diferentes módulos en los que se divide la función *full_representation.m*.

4.1.1 Time

Se comienza definiendo el intervalo de tiempos en que se representarán los datos. Dicho valor se obtiene del parámetro pasado a la función *time_file*, el cual, como se ha comentado anteriormente, es necesario pasarlo siempre como argumento porque, de lo contrario, no se podrán representar los datos de ningún sensor al no definirse el eje de tiempos, que define la duración del ensayo y la frecuencia con la cual los datos son recogidos.

De este modo, con la función de Matlab *fopen* y pasándole como primer argumento el fichero *time_file* nombrado y como segundo argumento *'r'* lo cual quiere decir que se está abriendo el fichero en modo lectura (reading) y pasando su output a la variable *fichero.*

```
%% Time
% Se obtiene el tiempo de ejecución del ensayo para representar
las
% gráficas en dicho intervalo
fichero = fopen(time_file,'r');
```

Cabe comentar que la definición de los argumentos de las funciones de Matlab puede consultarse escribiendo en la ventana de comando *help* *, siendo * la función a consultar, apareciendo tanto una definición de la misma, los argumentos de entrada y salida que ofrece y algún ejemplo de uso. Adicionalmente, en la web de Matlab puede obtenerse esta información de forma más extensa [7].

Se prosigue tomando la variable *fichero*, que es el contenido del fichero *time_file*, el cual consta de una cadena de texto cuya segunda palabra contiene el tiempo final del ensayo, es por ello que se toma esta palabra y se convierte a formato numérico con la función *str2num* de Matlab para poder utilizarla como valor final del eje de tiempos de representación de las gráficas, *time_stop*.

```
% Se toman las dos primeras palabras de la cadena de caracteres
cadena_caracteres = fscanf(fichero,'%s',2);


% Se toma solo la parte que contiene el tiempo y se pasa a formato
numérico
% obteniendo el instante de tiempo en segundos donde se para la
ejecución
time_stop = str2num(cadena_caracteres(8:14));
```

4.1.2 Encoder

Se comienza definiendo el código del encoder, el cual delimita el eje de tiempos de las diferentes gráficas, ya que este dispositivo es el que muestra la velocidad del tren, siendo el último valor que toma cuando se detiene el tren. Se sigue esta lógica ya que si algún sensor toma algún dato adicional al de parada del tren no nos es necesario conocerlo, ya que la relevancia de este estudio radica en el estudio de la vía y el impacto de la misma sobre las ruedas del vehículo, lo que quiere decir durante el desplazamiento del mismo sobre la vía.

De este modo, al igual que se hace con todos los otros dispositivos, lo primero que se hace es la comprobación de la existencia de datos en el fichero, en este caso *encoder_file*, definiendo una función de ejecución de la representación de los datos si el argumento que se ha pasado es distinto de 0.

```
%% Encoder
% Comprobación de representación del encoder.
% Carga del fichero de salida del mismo, pasado como parámetro de
esta función.
% El resultado se pasa a la función encargada de representar los
datos obtenidos
if encoder_file ~= 0
    …
end
```

Una vez realizada dicha comprobación, dentro de esta, donde se han puesto los tres puntos

suspensivos se incluye el siguiente código, el cual incluye mediante la función de Matlab *load* el contenido del fichero que contiene los datos del encoder en la variable *encoder*. Como previamente al análisis se ha incluido el símbolo *%* en la primera fila de cabecera, el cual realizar al función de convertirla en comentarios, no presenta problemas en la carga de datos con la función *load*, cargando exclusivamente los valores numéricos procedentes del sensor. A continuación, se abre una ventana de representación de una gráfica con la función figure. Una vez hecho esto, se utilizan las funciones parejas *hold on* y *hold of*, siendo la primera la cual fija que todos los datos que se van a ser representados hasta que se llegue a *hold of* serán dibujados dentro de la figura abierta previamente y a la cual se ha titulado ENDODER con la función *title.* Entre este conjunto de funciones se ha incluido la función creada por mí misma *encoder_representation*, la cual se encuentra dentro del mismo fichero *full_representation* al final, de este modo, se hace una llamada a la misma pasándole los argumentos las variables *encoder* y *time_stop*, a las cuales se les ha asignado valores anteriormente.

De este modo, conseguimos que el código sea modular, unificando las funciones exclusivas de representación de datos en gráficos al final del fichero.

```
encoder = load(encoder_file);
figure
hold on
title('ENCODER')
encoder_representacion(encoder,time_stop);
hold off
```

Para una mejor comprensión del lector, se presenta el contenido de la función nombrada, *encoder_representation* a continuación, siguiendo la lógica de la compilación del código y no tanto de su estructura.

Se comienza definiendo y asignando valores a tres variables. La primera es la frecuencia de adquisición de datos por parte del encoder *f_adq*, la cual se define como la inversa del tiempo entre tomas sucesivas de datos, la cual está definida como 0.04 segundos. Las dos frecuencias restantes se explican a continuación.

Es necesario aplicar un filtro Butterworth a los datos del encoder para producir la respuesta más plana posible hasta la frecuencia de corte, es decir, la salida se mantiene constante casi hasta la frecuencia de corte, luego disminuye a razón de 20n dB por década, donde n es el número de polos del filtro [8], se ve que en la ayuda de Matlab se contempla una función para tal fin, *butter* [7]. Como argumentos de esta función se encuentra el orden del filtro paso bajo y la mitad de la frecuencia de adquisición, es por ello que se crean las otras dos variables *f_nyq*, que representa la mitad de la frecuencia de adquisición *f_adq* y *f_c*, que presenta el orden del filtro, que en este caso se considera de orden 1, el más simple.

Como salidas de la función *butter*, se obtienen las variable *b* y *a*, las cuales se corresponden con los vectores denominador y numerador del filtro Butterworth con longitud n+1, en este caso 2. Dichos vectores son pasados como argumentos de entrada a la función de Matlab *filtfilt* junto a la columna de datos del encoder para realizarle el filtrado Butterworth a los datos, ya que la función es definida por Matlab como filtrado IIR digital de fase cero hacia adelante y hacia atrás. La salida de la misma son los datos filtrados, a los cuales se les ha denominado en un variable como *OMega.*

```
function encoder_representacion(encoder,time_stop)

f_adq = 1/0.04;
f_nyq = f_adq/2;
f_c = 1;

[b,a] = butter(4,f_c/f_nyq,'low');
OMega = filtfilt(b,a,encoder(:,2));

 …

end
```

Tras el filtrado de los datos, finalmente se representan los datos. Donde se incluyen los tres puntos suspensivos en el código anterior, se definen las funciones de representación de datos que se exponen en el siguiente recuadro. Se representan en una misma gráfica los datos recogidos por el encoder tanto en bruto, tal y como han sido recogidos, como tras aplicarles el filtro Butterworth, para poder identificar si los datos han sido recogidos correctamente y para poder ver su tendencia. Para una mejor visualización, se representan ambos datos en diferentes colores, de este modo, será fácilmente distinguibles ambas representaciones.

Para ello, se ha empleado la función *plot* de Matlab, a la cual se le pasan como argumentos de entrada, en primer lugar el eje horizontal, que se corresponde con la quinta columna del encoder, que es el eje de tiempos, como segundo argumento se pasa el valor de los datos recogidos por el encoder, ya sean sin filtrar recogidos en la segunda columna del archivo del encoder o filtrados como *OMega*. Cabe comentar que, para recoger los datos de una columna completa de una matriz, basta con indicar vector(:, n_columna), correspondiendo los dos puntos con todas las filas y n_columna con el número de la columna, correspondiendo la primera con un 1 y así progresivamente. Como tercer argumento a la función plot se le pasa un carácter correspondiente al color en el que se desea que se represente dicha curva, se han elegido *'b'* azul (blue) y *'r'* rojo (red), correspondiendo el azul con los datos en bruto y el rojo con los datos filtrados, el cual al ponerse en el segundo plot, se sobrepone al azul, siendo claramente diferenciables uno sobre el otro.

Adicionalmente, se utilizan las funciones *xlabel*, *ylabel* y *xlim* para customizar los ejes de representación, correspondiendo los dos primeros al nombramiento de los dos ejes, x e y. Como puntualización, se ha buscado la nomenclatura homóloga para poder representar la letra griega omega en el eje y, la cual se corresponde con *\omega_m* siendo así más visual y siguiendo así la nomenclatura ingenieril para representar la velocidad angular. Por su parte, *xlim* delimita el rango del eje de tiempos x, desde cero hasta el *time_stop* correspondiente.

```
plot(encoder(:,5)*10^-3,encoder(:,2),'b')
plot(encoder(:,5)*10^-3,OMega,'r')
xlabel('Time (s)'),ylabel('\omega_m (rpm)'),xlim([0 time_stop])
```

4.1.3 IMUs

Del mismo que se ha hecho con el encoder, se realiza el análisis del código de representación de datos de los tres sensores IMUs utilizadas para la recogida de los datos de la dinámica vehicular del tren a escala.

En este caso se dispone de dispositivos con generación de datos del mismo formato, así que se programan de la misma forma denominándolos de forma diferente como *imu1, imu2* e *imu3*, para que no exista solapamiento entre los datos y sea fácilmente comprensible.

Así pues, se distinguen tres bloques, uno para cada IMU, en los cuales lo primero que se hace es la comprobación de la existencia de datos en el fichero, es decir, si el argumento que se ha pasado es distinto de 0. De esta forma, se pueden representar los valores de cada IMU de forma individual, sin necesidad de que los tres ficheros presenten datos o se puede analizar los datos correspondientes a una o dos IMUs en concreto si se requiere, lo cual no será permitido cuando se automaticen los datos en el siguiente capítulo (*auto_full_representation)* de este proyecto, he aquí el mayor potencial de esta función *full_representation.*

Como se comentó anteriormente, para realizar el código de la forma más eficiente y siguiendo las buenas prácticas de la programación, al igual que se hizo con el encoder, la función de representación de los datos de la IMU se define en otra función denominada *imu_representacion*.

La gran ventaja que aporta este estructura es que cada bloque correspondiente a cada una de las tres IMUs hará una llamada a dicha función para representar los datos, siendo no necesario la replicación en tres veces del mismo código para cada IMU, suponiendo un ahorro en tiempo de computación y acortando la magnitud del código.

Los argumentos necesarios para dicha función serán, al igual que anteriormente, el contenido del fichero de la IMU en cuestión el cual se ha pasado a la variable correspondiente mediante la función *load* y el tiempo final de representación obteniendo anteriormente del fichero *time_file,* que es el *time_stop*, el cual variará por cada ensayo, pero no entre los diferentes sensores, teniendo todos el mismo valor de final de tiempo de representación. Como previamente al análisis se ha incluido el símbolo *%* en la primera fila de cabecera, el cual realizar al función de convertirla en comentarios, no presenta problemas en la carga de datos con la función *load*, cargando exclusivamente los valores numéricos procedentes del sensor.

```
%% IMUs

% Comprobación de representación de la IMU 1.
% Carga del fichero de salida de la misma, pasado como parámetro
de esta función.
% El resultado se pasa a la función encargada de representar los
datos obtenidos.
if imu1_file ~= 0
    imu1 = load(imu1_file);
    figure
    hold on
    title('IMU 1')
    imu_representacion(imu1,time_stop);
    hold off
end
```

```
% Ídem para IMU 2.
if imu2_file ~= 0
    imu2 = load(imu2_file);
    figure
    hold on
    title('IMU 2')
    imu_representacion(imu2,time_stop);
    hold off
end

% Ídem para IMU 3.
if imu3_file ~= 0
    imu3 = load(imu3_file);
    figure
    hold on
    title('IMU 3')
    imu_representacion(imu3,time_stop);
    hold off
end
```

A continuación, se presenta el contenido de la función *imu_representation,* la cual se encuentra en el bloque final del fichero *full_representation* pero es mejor presentarlo en este momento para una mejor comprensión del lector, ya que es llamada en el bloque anterior.

En esta ocasión no se realiza un filtrado de los datos como se hizo con el encoder, ya que el filtrado de estos datos espúreos se desarrolla en un capítulo posterior en este proyecto.

Ahora se procede a representar los seis valores recogidos por cada IMU, los cuales corresponden con la velocidad angular en cada uno de los tres ejes (x,y,z) y la aceleración en cada uno de los tres ejes (x,y,z). Para que el analista pueda verlos de mejor forma, se representan las seis gráficas en una misma ventana, lo cual se consigue haciendo uso de la función *subplot*, indicándole tres argumentos, número total de filas, número total de columnas y número de posición de la gráfica en cuestión, realizando el barrido de arriba abajo y de izquierda a derecha.

Como puntualización, cabe decir que, al encontrarse los datos de tiempo de recogida de datos en microsegundos, los cuales se encuentran en la primera columna del fichero de cada IU, estos se multiplican por 10^{-6} para su representación en segundos.

```
function imu_representacion(imu,time_stop)

subplot(3,2,1)
plot(imu(:,1)*10^-6,imu(:,2))
title('Angular rate (deg/s)')
xlabel('Time (s)'), ylabel('Angular rate X (deg/s)'),xlim([0
time_stop])

subplot(3,2,3)
plot(imu(:,1)*10^-6,imu(:,3))
xlabel('Time (s)'), ylabel('Angular rate Y (deg/s)'),xlim([0
time_stop])
```

```
subplot(3,2,5)
plot(imu(:,1)*10^-6,imu(:,4))
xlabel('Time (s)'), ylabel('Angular rate Z (deg/s)'),xlim([0
time_stop])

subplot(3,2,2)
plot(imu(:,1)*10^-6,imu(:,5))
title('Acceleration (g)')
xlabel('Time (s)'), ylabel('Acceleration X (g)'),xlim([0
time_stop])

subplot(3,2,4)
plot(imu(:,1)*10^-6,imu(:,6))
xlabel('Time (s)'), ylabel('Acceleration Y (g)'),xlim([0
time_stop])

subplot(3,2,6)
plot(imu(:,1)*10^-6,imu(:,7))
xlabel('Time (s)'), ylabel('Acceleration Z (g)'),xlim([0
time_stop])

end
```

Adicionalmente, se ha contemplado que los tres ficheros de las IMUs tengan datos, por lo que se añade unas representaciones adicionales a las expuestas anteriormente si se da esta situación. Estas representaciones consisten en representar las señales a estudiar de velocidades angulares y aceleraciones de las tres IMUs en una misma gráfica, solapando los datos entre ellos de una forma lógica para que se pueda analizar una comparativa entre los tres sensores. De este modo, puede ver qué IMU de las tres presenta más desviaciones y realizar un estudio más fácil sin tener que cambiar de pantallas de representación y mirar datos sobre cada gráfica individual y que sea el analista quien ha de hacer el solape de los datos manualmente entre las diferentes gráficas.

Para realizar dicha representación se ha creado el siguiente código, cuya primera comprobación es que los ficheros de los tres sensores inerciales presentan datos y no se han pasado como argumento 0 ninguno de ellos.

Se presentan tanto las velocidades angulares como las aceleraciones cartesianas de forma individual para cada eje en una gráfica, haciendo solape de los valores de cada IMU, representando con color azul a la IMU 1, la cual se sitúa en el tren en el *wheelset*, la IMU 3 en color rojo correspondiente a la que se sitúa en el *bogie* y de color verde a la IMU 2 situada en el *carbody*.

Esta sucesión se ha seguido de forma lógica ya que, tras analizar la IMU que presentaba más desviaciones era la IMU situada en el wheelset y la que presenta menos es la situada en el carbody, de este modo se pueden ver correctamente las tres curva, una sobre otra.

Al igual que con todas la gráficas, se han denominado el nombre deseado al eje de tiempos y al eje de y con sus correspondientes unidades de medida, añadiendo la funcionalidad en esta ocasión de subíndices para una visualización más ingenieril.

Adicionalmente, se ha añadido en esta ocasión una leyenda para mostrar al analista a qué IMU corresponde cada una de las curva, denominando su localización.

```
% Representación conjunta de las 3 IMUs
if imu1_file ~= 0
    if imu2_file ~= 0
        if imu3_file ~= 0
            % Angular velocities
            figure, plot(imu1(:,1)*10^-
6,imu1(:,2),'b',imu3(:,1)*10^-6,imu3(:,2),'r',imu2(:,1)*10^-
6,imu2(:,2),'g'), title('Angular Velocity \omega_x'), xlabel('Time
(s)'), ylabel('\omega_x (deg/s)'),
legend('wheelset','bogie','carbody'),xlim([0 time_stop])
            figure, plot(imu1(:,1)*10^-
6,imu1(:,3),'b',imu3(:,1)*10^-6,imu3(:,3),'r',imu2(:,1)*10^-
6,imu2(:,3),'g'), title('Angular Velocity \omega_y'), xlabel('Time
(s)'), ylabel('\omega_y (deg/s)'),
legend('wheelset','bogie','carbody'),xlim([0 time_stop])
            figure, plot(imu1(:,1)*10^-
6,imu1(:,4),'b',imu3(:,1)*10^-6,imu3(:,4),'r',imu2(:,1)*10^-
6,imu2(:,4),'g'), title('Angular Velocity \omega_z'), xlabel('Time
(s)'), ylabel('\omega_z (deg/s)'),
legend('wheelset','bogie','carbody'),xlim([0 time_stop])

            % Cartesian accelerations
            figure, plot(imu1(:,1)*10^-
6,imu1(:,5),'b',imu3(:,1)*10^-6,imu3(:,5),'r',imu2(:,1)*10^-
6,imu2(:,5),'g'), title('X-acceleration a_x'), xlabel('Time (s)'),
ylabel('a_x (g)'), legend('wheelset','bogie','carbody'),xlim([0
time_stop])
            figure, plot(imu1(:,1)*10^-
6,imu1(:,6),'b',imu3(:,1)*10^-6,imu3(:,6),'r',imu2(:,1)*10^-
6,imu2(:,6),'g'), title('Y-acceleration a_y'), xlabel('Time (s)'),
ylabel('a_y (g)'), legend('wheelset','bogie','carbody'),xlim([0
time_stop])
            figure, plot(imu1(:,1)*10^-
6,imu1(:,7),'b',imu3(:,1)*10^-6,imu3(:,7),'r',imu2(:,1)*10^-
6,imu2(:,7),'g'), title('Z-acceleration a_z'), xlabel('Time (s)'),
ylabel('a_z (g)'), legend('wheelset','bogie','carbody'),xlim([0
time_stop])

        end
    end
end
```

4.1.4 Láseres de triangulación

Al igual que se ha realizado con los sensores inerciales anteriores, se procede a la exposición y explicación del código realizado en Matlab para la representación de los datos recogidos por los láseres medidores de distancia de alta precisión para el estudio de las oscilaciones que presenta el eje de la rueda a su paso por la vía ferroviaria.

En esta ocasión también se dispone de dos dispositivos con generación de paquetes de datos en el mismo formato, así que se programan de la misma forma denominándolos esta vez como *laser1* y *laser2*.

Se distinguen dos bloques, uno para cada sensor de distancia, en los cuales lo primero que se hace es la comprobación de la existencia de datos en el fichero, es decir, si el argumento que se ha pasado es distinto de 0. De esta forma, se representan los valores de cada láser de forma individual.

Al igual que se han programado los sensores anteriores, la función de representación de los datos del sensor se define en otra función denominada *laser_representacion*.

Como ocurría anteriormente con los sensores inerciales, la gran ventaja que aporta este estructura es que cada bloque correspondiente a cada uno de los dos láser hará una llamada a dicha función para representar los datos, no siendo necesario la replicación del mismo código para cada láser.

Los argumentos de entrada de dicha función son, al igual que anteriormente, los datos contenidos en el fichero del sensor medidor de distancias, pasados a la variable correspondiente mediante la función *load,* y el tiempo final de representación obteniendo anteriormente del fichero *time_file,* que es *time_stop*.

```
%% Lasers

% Comprobación de representación de laser 1.
% Carga del fichero de salida del mismo, pasado como parámetro de
esta función.
% El resultado se pasa a la función encargada de representar los
datos obtenidos, se puede consultar más abajo en este mismo
fichero.
if laser1_file ~= 0
    laser1 = load(laser1_file);
    figure
    hold on
    title('LASER 1')
    laser_representacion(laser1,time_stop);
    hold off
end

% Ídem para laser 2.
if laser2_file ~= 0
    laser2 = load(laser2_file);
    figure
    hold on
    title('LASER 2')
    laser_representacion(laser2,time_stop);
    hold off
end
```

A continuación, se presenta el contenido de la función *laser_representation,* la cual es llamada en el bloque anterior y se encuentra definida en el final del fichero *full_representation*.

Los valores de tiempo se encuentran en microsegundos, así que se multiplican los valores por 10^{-6}.

```
function laser_representacion(laser,time_stop)
  plot(laser(:,1)*10^-6,laser(:,2))
  xlabel('Time (s)'), ylabel('Distance (mm)'),xlim([0 time_stop])
end
```

Adicionalmente, se ha contemplado que los ficheros de ambos sensores medidores de distancias contengan tengan datos, así que se añaden unas visualizaciones de dichos datos de forma conjunta para un mejor análisis de los mismos en una misma ventana, haciendo uso de la función *subplot* ya presentada.

Para realizar dicha representación se ha creado el siguiente código, cuya primera comprobación es que los ficheros de los dos sensores no se pasan como argumento 0 a la función *full_representation*.

En esta ocasión, se presentan en una ventana tres gráficas organizadas de forma vertical, mostrando la de la parte más superior los valores recogidos por el láser situado en el carbody, la del centro los valores recogidos por el láser situado en el bogie y, en la parte inferior, esta tercera gráfica mostrando los valores de ambos sensores solapados, sobreponiendo los valores del láser situado en el bogie, en rojo, a los del láser situado en el carbody, azul, ya que este presenta mayor desviaciones según se ha analizado tras varias ejecuciones del código.

Al igual que con todas la gráficas, se han denominado el nombre deseado al eje de tiempos y al eje de y con sus correspondientes unidades de medida. Adicionalmente, se ha añadido en esta ocasión una leyenda en la última gráfica exclusivamente para la correcta denominación de las curvas de ambos sensores.

```
% Representación conjunta de los 2 láseres
% Comprobación de si se han pasado como parámetro los dos ficheros
if laser1_file ~= 0
    if laser2_file ~= 0
            % Distance lasers
            figure, subplot(3,1,1), plot(laser1(:,1)*10^-
6,laser1(:,2),'b'), xlabel('Time (s)'), ylabel('z (mm)'),
title('Carbody distance laser'),xlim([0 time_stop])
            subplot(3,1,2), plot(laser2(:,1)*10^-
6,laser2(:,2),'r'), xlabel('Time(s)'), ylabel('z (mm)'),
title('Bogie distance laser'),xlim([0 time_stop])
            subplot(3,1,3), plot(laser1(:,1)*10^-6,laser1(:,2)-
laser1(1,2),'b',laser2(:,1)*10^-6,laser2(:,2)-laser2(1,2),'r'),
title('Relative Distance lasers \Deltaz (mm)'), xlabel('Time
(s)'), ylabel('\Deltaz (mm)'), legend('Carbody laser','Bogie
laser'),,xlim([0 time_stop])
    end
end
```

4.2 Automatización de post-procesamiento de datos

En el apartado anterior se muestra la función *full_representation.m* creada para la representación de los datos de los diferentes sensores de forma individual, la mayor condición restrictiva para el correcto funcionamiento del código es que es necesario incluir, como último argumento, el fichero que define el tiempo de ejecución del ensayo en cuestión.

Como mejora, no contemplada en primera instancia en el proyecto pero que es un pilar fundamental del mismo, la gran ayuda que aporta la automatización del proceso de ejecución de representación de datos, es decir, sin necesidad de pasarle cada uno de los ficheros de datos obtenidos por los sensores si todos ellos precisan de contenido. La principal motivación para ello es la aceleración del proceso del análisis de la correcta ejecución de la toma de datos in-situ al realizar las pruebas en campo, segundos después de realizar el ensayo, sin necesidad de apenas tener que escribir nada en la ventana de comandos de Matlab para la ejecución del mismo, simplemente es necesario abrir la función de forma previa y pegar la localización donde se encuentran los documentos a analizar.

A continuación, se explican los diferentes módulos que se han creado en Matlab para la automatización del fichero *full_representation.m*, denominándose el nuevo fichero como *auto_full_representation.m*.

La primera parte de la función con mismo nombre del fichero, *auto_full_representation*, es donde radica la gran diferencia respecto al fichero anterior, ya que es donde se procede a la carga de ficheros de datos en lugar de pasarlos como argumentos de la función. Para ello, se declara la variable *directorio* a la cual se le asigna el PATH donde se encuentran situados todos los ficheros. Es preciso que este precedido y finalice con el carácter ‘\’ para que dicha variable sea una cadena de caracteres, es decir, una variable de tipo *string*. Como requisito de buenas prácticas, ha de existir una carpeta por ensayo, es decir, sólo un fichero por cada sensor en dicho directorio correspondiente a un ensayo en exclusiva.

Se continúa haciendo uso de la función de Matlab *strcat*, la cual unifica las dos cadenas de caracteres que se le pasan como argumentos. En esta ocasión se han concatenado la cadena de caracteres del PATH pasado a la variable directorio y **.csv*, de este modo, al aplicar la función de Matlab *dir* a dicha cadena de caracteres, busca todos los documentos que se encuentran dentro de ese directorio con la extensión de fichero csv (los cuales se corresponden a los datos que devuelven los sensores) y los pasa a la variable estructura *d*. Para obtener los nombres de dichos ficheros, se emplea la nomenclatura de las variables estructuras para acceder a una de sus propiedades, *{d.name}*. De este modo, se pasan los nombres de los dicheros al vector *names_files*.

```
function auto_full_representation
close all


%% Load files

% Se asignan los archivos de salida de los distintos componentes
en formato .csv
directorio = 'ensayos_alamillo_28_9_17\2\';
d = dir(strcat(directorio,'*.csv'));
names_files = {d.name};
```

A continuación, se procede a introducir los datos de cada uno de los ficheros dentro de sus variables correspondientes con la función *load*, al igual que se realizó anteriormente en la función *full_representation.* Previamente a ello, es necesario obtener el fichero comentado, los cuales se encuentran dentro del directorio, así que se concatena con la función *strcat* tanto dicho array *directorio* como el nombre del fichero de la variable *name_file*, al cual accede denominando de la posición dentro del vector, siguiendo el orden en el que han sido guardados: encoder, imu1, imu2, laser1, imu3 y laser2.

```
encoder_file = strcat(directorio,char(names_files(1)));
encoder = csvread(encoder_file,1,0);
imu1_file = strcat(directorio,char(names_files(2)));
imu1 = load(imu1_file);
imu2_file = strcat(directorio,char(names_files(3)));
imu2 = load(imu2_file);
laser1_file = strcat(directorio,char(names_files(4)));
laser1 = load(laser1_file);
imu3_file = strcat(directorio,char(names_files(5)));
imu3 = load(imu3_file);
laser2_file = strcat(directorio,char(names_files(6)));
laser2 = load(laser2_file);
```

4.2.1 Encoder

Como se ha podido observar, en esta ocasión no se toma el valor de tiempo final de la ejecución del ensayo que se obtenía del fichero time_file. De lo contrario, se considera como tiempo final de la ejecución del ensayo el tiempo en el que el encoder toma el último valor del ensayo, es decir, hasta que el encoder deja de recibir información del tren porque se le ha dado la señal para que finalice el ensayo. De este modo, no es necesario que se genere el fichero de tiempos, agilizando el procesado de datos. Dicho valor es tomado indicando que se tome el último valor de la quinta columna de la variable que contiene los datos del encoder, *encoder(end,5)*. Así mismo, este valor es multiplicado por 10^{-3} ya que los tiempos presentados en el fichero del encoder se encuentran en milisegundos. Del mismo que en el otro fichero, se llama a la función que representa los datos, la cual se encuentra al final del fichero.

```
%% Encoder

% Carga del fichero de salida del mismo
% El resultado se pasa a la función encargada de representar los
datos obtenidos
if encoder_file ~= 0
    % Se toma como tiempo límite de la simulación el último
instante de tiempo de toma de datos por parte del encoder
    time_stop = encoder(end,5)*10^-3;
    figure, hold on
    encoder_representacion(encoder,time_stop);
    hold off
end
```

Esta función de representación de datos recogidos por el encoder no es exactamente igual que la anterior considerada en la función *full_representation*, ya que se añade la representación adicional de la distancia recorrida por el tren en función del tiempo de ejecución del ensayo. Es por ello que se le pasa como parámetros iniciales el radio de la rueda *R* y el tiempo de muestro del sistema de adquisición *t_m*, el cual es pasado para calcular la frecuencia de muestreo del sistema *f_adq*, que a su vez se utiliza para calcula la frecuencia de Nyquist requerida como parámetro por el filtro de Butterworth [8]. Del mismo modo, se indica que el número de dientes del encoder *d_enc*.

Se comienza obteniendo los instantes de tiempo de adquisición de datos en segundos, tomando la quinta columna del fichero de datos del encoder y pasándosela a la variable *time*.

```
%% Función de representación encoder

function encoder_representacion(encoder,time_stop)

R = 0.0635;       % radio de la rueda
t_m = 0.008;      % tiempo de muestreo del sistemas de adquisición
8 ms, (seg)
f_adq = 1/t_m;    % frecuencia de muestreo del sistemas de
adquisición
f_nyq = f_adq/2; % tiempo de muestreo de Nyquist al doble del t_m
f_c = 1;
d_enc = 1500;     % dientes del encoder

time = encoder(:,5)*10^-3;     % instantes de tiempo de adquisición
de datos (seg)
```

Se prosigue calculando la velocidad en metros/segundos, tomando la cuarta columna del encoder que muestra la velocidad angular exclusivamente. Así que se realiza el cálculo:

$$velocidad\ (^{m}/_{s}) = \frac{2 \cdot \pi \cdot \frac{\omega_{enc}\ (rpm)}{dientes_{enc}}}{t_{muestreo} \cdot Radio_{rueda\ tren}(m)} \tag{4–1}$$

El cual es programado del siguiente modo:

```
% Se calcula la velocidad
vel = 2*pi*(encoder(:,4)/d_enc)/t_m*R;
```

Se prosigue aplicando el filtro de Butterworth, *butter*, ya comentado para el fichero *full_representation* a los nuevos datos de la velocidad mediante la función *filtfilt*.

```
% Se filtra la señal velocidad con un filtro de Butterworth
[b,a] = butter(4,f_c/f_nyq,'low');
OMega = filtfilt(b,a,vel);
```

A continuación, se prosigue calculando el espacio recorrido por el tren durante el ensayo del siguiente modo:

$$velocidad\ (^{m}/_{s}) = Radio_{rueda\ tren}(m) \cdot posición_{enc} \cdot \frac{2 \cdot \pi}{dientes_{enc}} \qquad (4\text{–}2)$$

El cual es programado del siguiente modo:

```
% Se calcula el espacio recorrido
s = R*encoder(:,3)*2*pi/d_enc;
```

A continuación, se crea el siguiente código para asegurar que los datos calculados de distancia recorrida es la correcta, tomando los intervalos de tiempos adecuados. Así se podrá realizar una comparativa con el valor del espacio calculado anteriormente. Se crea para ello una nueva variable *s_acu*, acrónimo de space accuracy, la cual se inicializa dándole un vector de ceros de longitud igual al tiempo de ejecución. Se toma como condición para el primer valor que el espacio será el instante de muestreo por la velocidad en el primer instante. Para los instantes sucesivos, estos se reproducirán a partir del espacio recorrido justo anterior *s_acu(i-1)* más el espacio recorrido a dicha velocidad *vel(i)* en el instante de muestreo *t_m*.

```
% Comprobación de que el espacio calculado entre instantes de
tiempo será correcto
s_acu = zeros(1,length(time)-1);
for i = 1:length(time)
    if i == 1
        s_acu(i) = t_m*vel(i);
    else
        s_acu(i) = s_acu(i-1) + t_m*vel(i);
    end
end
```

La función *encoder_representation* finaliza con la propia representación de los datos calculados anteriormente, haciendo uso de la función *subplot* para representar dos gráficos en una misma ventana. El primero de ellos, en la parte superior, mostrará de forma conjunta tanto la velocidad teórica como la velocidad filtrada, mientras que el segundo gráfico de la parte inferior muestra el espacio recorrido por el tren y el espacio teórico calculado anteriormente.

```
% Representación de la salida del encoder, velocidad y espacio
recorrido
subplot(2,1,1)
plot(time,vel,'r', time,OMega,'b')
title('Motor encoder')
legend('Real velocity','Filtered velocity')
xlabel('Time (s)'),ylabel('Velocity (m/s)'),xlim([0 time_stop])
```

```
subplot(2,1,2)
plot(time,s,'r', time,s_acu,'b')
legend('Real space','Calculated space')
xlabel('Time (s)'),ylabel('Space (m)'),xlim([0 time_stop])

end
```

4.2.2 IMUs

Por su parte, los sensores inerciales se van a representar de forma conjunta como aparecía en el fichero *full_representation*, seis gráficas dentro de una misma ventana, cada una de ellas representando uno de los seis parámetros obtenidos por los sensores, solapándose los valores de los tres sensores en cada gráfica para poder realizar una comparativa a simple vista por parte del analista. Cabe comentar que las representaciones individuales de los sensores se han omitido en este fichero ya que se ha considerado, por parte de los analistas, que la gráfica conjunta mostrada aporta todos los datos deseados de los ensayos. Para ello, como ya se ha comentado, se hace uso de la función *subplot*, para indicar la localización de cada gráfica en la ventana, y la función *plot* representa los diferentes valores de las variables deseadas en función del tiempo. En esta ocasión no se ha precisado la creación de una función a parte exclusivamente para la representación.

```
%% IMUs

% Representación conjunta de las 3 IMUs
if length(imu1_file) ~= 0
    if length(imu2_file) ~= 0
        if length(imu3_file) ~= 0
            % Angular velocities
            figure, subplot(3,2,1), plot(imu1(:,1)*10^-
6,imu1(:,2),'b',imu3(:,1)*10^-6,imu3(:,2),'r',imu2(:,1)*10^-
6,imu2(:,2),'g'), title('Angular Velocity \omega_x'), xlabel('Time
(s)'), ylabel('\omega_x (deg/s)'),
legend('wheelset','bogie','carbody')
            xlim([0 time_stop])
            ylim([-5,5])
            subplot(3,2,3), plot(imu1(:,1)*10^-
6,imu1(:,3),'b',imu3(:,1)*10^-6,imu3(:,3),'r',imu2(:,1)*10^-
6,imu2(:,3),'g'), title('Angular Velocity \omega_y'), xlabel('Time
(s)'), ylabel('\omega_y (deg/s)'),
legend('wheelset','bogie','carbody')
            xlim([0 time_stop])
            ylim([-3,3])
            subplot(3,2,5), plot(imu1(:,1)*10^-
6,imu1(:,4),'b',imu3(:,1)*10^-6,imu3(:,4),'r',imu2(:,1)*10^-
6,imu2(:,4),'g'), title('Angular Velocity \omega_z'), xlabel('Time
(s)'), ylabel('\omega_z (deg/s)'),
legend('wheelset','bogie','carbody')
            xlim([0 time_stop])
            ylim([-2,2])
```

```
            % Cartesian accelerations
            % Se resta el offset del segundo valor tomado porque
el primero contiene cierto error en algunas ocasiones
            subplot(3,2,2), plot(imu1(:,1)*10^-6,imu1(:,5)-
imu1(2,5),'b',imu3(:,1)*10^-6,imu3(:,5)-
imu3(2,5),'r',imu2(:,1)*10^-6,imu2(:,5)-imu2(2,5),'g'), title('X-
acceleration a_x'), xlabel('Time (s)'), ylabel('a_x (g)'),
legend('wheelset','bogie','carbody')
            xlim([0 time_stop])
            ylim([-5,5])
            subplot(3,2,4), plot(imu1(:,1)*10^-6,imu1(:,6)-
imu1(2,6),'b',imu3(:,1)*10^-6,imu3(:,6),'r',imu2(:,1)*10^-
6,imu2(:,6)-imu2(2,6),'g'), title('Y-acceleration a_y'),
xlabel('Time (s)'), ylabel('a_y (g)'),
legend('wheelset','bogie','carbody')
            xlim([0 time_stop])
            ylim([-10,10])
            subplot(3,2,6), plot(imu1(:,1)*10^-6,imu1(:,7)-
imu1(2,7),'b',imu3(:,1)*10^-6,imu3(:,7)-
imu3(2,7),'r',imu2(:,1)*10^-6,imu2(:,7)-imu2(2,7),'g'), title('Z-
acceleration a_z'), xlabel('Time (s)'), ylabel('a_z (g)'),
legend('wheelset','bogie','carbody')
            xlim([0 time_stop])
            ylim([-20,20])
        end
    end
end
```

4.2.3 Láseres de triangulación

Al igual que se ha realizado con las IMUs la representación de los datos los sensores de distancia es la misma que en el fichero *full_representation*. En esta ocasión, al igual que se ha realizado con los sensores inerciales, no se comprueba si los ficheros presentan un valor diferente de cero, sino si la longitud de datos contenidos en el documento es distinta de cero, lo cual quiere decir que no están vacíos. Del mismo modo, se presentan en una ventana tres gráficas organizadas de forma vertical, mostrando la de la parte más superior los valores recogidos por el láser situado en el carbody, la del centro los valores recogidos por el láser situado en el bogie y, en la parte inferior, esta tercera gráfica mostrando los valores de ambos sensores solapados, sobreponiendo los valores del láser situado en el bogie, en rojo, a los del láser situado en el carbody, azul, ya que este presenta mayor desviaciones según se ha analizado tras varias ejecuciones del código.

Al igual que con todas la gráficas, se han denominado el nombre deseado al eje de tiempos y al eje de y con sus correspondientes unidades de medida. Adicionalmente, se ha añadido en esta ocasión una leyenda en la última gráfica exclusivamente para la correcta denominación de las curvas de ambos sensores.

```
%% Lasers

% Comprobación de representación de la laser 1.
% Carga del fichero de salida del mismo, pasado como parámetro de
esta
% función.
% El resultado se pasa a la función encargada de representar los
datos
% obtenidos, se puede consultar más abajo en este mismo fichero.
if length(laser1_file) ~= 0
    figure, subplot(3,1,1), plot(laser1(:,1)*10^-
6,laser1(:,2),'b'), xlabel('Time (s)'), ylabel('Distance (mm)'),
title('Carbody distance laser')
    xlim([0 time_stop])
    ylim([80,87])
end

% Ídem para laser 2.
if length(laser2_file) ~= 0
    subplot(3,1,2), plot(laser2(:,1)*10^-6,laser2(:,2),'r'),
xlabel('Time(s)'), ylabel('Distance (mm)'), title('Bogie distance
laser')
    xlim([0 time_stop])
    ylim([10,16])
end

% Representación conjunta de los 2 lásers
if length(laser1_file) ~= 0
    if length(laser2_file) ~= 0
            % Distance lasers
            subplot(3,1,3), plot(laser1(:,1)*10^-6,laser1(:,2)-
laser1(2,2),'b',laser2(:,1)*10^-6,laser2(:,2)-laser2(2,2),'r'),
title('Relative Distance lasers \Deltaz (mm)'), xlabel('Time
(s)'), ylabel('\Deltaz (mm)'), legend('Carbody laser','Bogie
laser'),xlim([0 time_stop])
            ylim([-5,5])
    end
end
```

4.3 Corrección de datos espúreos

En los ficheros de representación anteriores, tanto *full_representation.m* como *auto_full_representation.m* los datos mostrados son los procedentes de los sensores en formato bruto a excepción de la aplicación del filtro Butterworth al encoder. Con el análisis de algunos ensayos precedentes en el tiempo se observaba la presencia de datos espúreos en la representación de los mismos, los cuales no eran representativos para el análisis que se deseaba realizar ya que presencia era de forma extraordinaria y muy poco frecuente. Sin embargo, las gráficas podían mostrar grandes desviaciones respecto a los datos de mayor frecuencia ya que estos valores espúreos presentaban extremadamente elevados.

Es por ello que el análisis de la existencia de estos datos y la eliminación de los mismos es objetivo de este proyecto. Como primer pensamiento, se puede considerar que delimitando los ejes de representación de las diferentes gráficas, estos valores extremos no aportarán perturbación a la representación, sin embargo, como la programación de este post-procesamiento de señales se ha realizado para que presente total adaptabilidad a los diferentes escenarios a los que se podría exponer el vehículo, automatizando todos los procesos de representación, se ha descartado esta fácil programación exclusivamente de delimitación de límites.

Así pues, para tal objetivo se ha desarrollado en lenguaje m en Matlab una archivo denominado *auto_full_representation_filtered.m*, el cual toma como referencia el fichero *auto_full_representation.m* y se le añade la funcionalidad de eliminación de estos picos para mostrar los datos de forma representativa.

Como se puede ver a continuación, la mayor diferencia respecto a la carga de datos de *auto_full_representation.m* radica en que, en *auto_full_representation_filtered.m*, se han denominado a las variables que contienen los datos de los ficheros con el sufijo **_peaks*, de este modo, se puede distinguir fácilmente si los datos han sido filtrados de datos espúreos o no.

```
function auto_full_representation_filtered
close all


%% Load files

% Se asignan los archivos de salida de los distintos componentes
en formato .csv
directorio = 'ensayos_alamillo_28_9_17\2\';
d = dir(strcat(directorio,'*.csv'));
names_files = {d.name};

encoder_file = strcat(directorio,char(names_files(1)));
encoder = csvread(encoder_file,1,0);
imu1_file = strcat(directorio,char(names_files(2)));
imu1_peaks = load(imu1_file);
imu2_file = strcat(directorio,char(names_files(3)));
imu2_peaks = load(imu2_file);
laser1_file = strcat(directorio,char(names_files(4)));
laser1_peaks = load(laser1_file);
imu3_file = strcat(directorio,char(names_files(5)));
imu3_peaks = load(imu3_file);
laser2_file = strcat(directorio,char(names_files(6)));
laser2_peaks = load(laser2_file);
```

En este fichero, el valor final del eje de tiempos de representación de las diferentes gráficas, se toma como el valor mínimo de todos los tiempos finales de toma de datos de los diferentes sensores, es decir, del encoder, las tres IMUs y los dos láseres, haciendo uso de la función de Matlab *min*.

Para que el usuario tenga conocimiento de ello, se representa por la ventana de comandos de Matlab el sensor referenciado por: 1-encoder, 2-imu1, 3-imu2, 4-imu3, 5-laser1 y 6-laser2.

```
%% Se busca el instante de tiempo final más bajo de todos los
componentes en la toma de datos, pasados a segundos
stop_times = [encoder(end,5)*10^-3, imu1_peaks(end,1)*10^-6,
imu2_peaks(end,1)*10^-6, laser1_peaks(end,1)*10^-6,
imu3_peaks(end,1)*10^-6, laser2_peaks(end,1)*10^-6];
[time_stop,i] = min(stop_times);

% Para saber qué componente ha sido, se muestra por la línea de
comandos para nuestra información
componentes = [1,2,3,4,5,6];
fprintf('Componente que ha cortado la toma de datos el primero:
%i\n',componentes(i));
```

Ahora se procede al filtrado de dichos picos, para ello se ha creado la función *peaks_filter*. Para su empleo se le pasan como argumentos de entradas todas las variables que contienen los datos de los sensores en bruto, a excepción del encoder, además del tiempo de parada *time_stop* definido anteriormente Por su parte, la función ofrece como salida cinco variables que contienen los datos de los sensores filtrados de picos esporádicos, es por ello que se denominan del mismo modo pero quitando el sufijo **_peaks*.

```
%% Procesado de ficheros, filtrado de picos
[imu1,imu2,imu3,laser1,laser2] =
peaks_filter(imu1_peaks,imu2_peaks,imu3_peaks,laser1_peaks,laser2_
peaks,time_stop);
```

A continuación, se presenta la definición de la función *peaks_filter*. Esta función está dividida en dos módulos de filtrados, ya que el dato extremo puede darse en el eje del tiempo o en el eje de los datos tomados por los sensores. En ambos casos se toma un valor excepcional, por encima del cual se considera un valor extremo, el cual se ha de eliminar. Para el caso del tiempo, se toma como valor excepcional 0,05 s más del tiempo de parada del ensayo, ya que sólo se tomarán los datos correspondientes al tiempo del ensayo. Para el caso de valores de salida de sensores, se toma como valor excepcional 20, el cual se comentará más adelante.

```
%% Función de procesado de ficheros, ausencia de picos

function [imu1,imu2,imu3,laser1,laser2] =
peaks_filter(imu1,imu2,imu3,laser1,laser2,time_stop)

k = 1;  % Para índices

%% Filtrados según el tiempo
 valor_excep_time = time_stop + 0.05;
 …

%% Filtrados según el valor de salida
 valor_excepcional = 20;
 …
end
```

Donde se han incluido anteriormente puntos suspensivos para que se viera la estructura, se presenta ahora el código que ahí se encuentra.

Se hace un filtrado por cada dispositivo, es decir, para cada una de las tres IMUs y para cada uno de los dos láseres medidores de distancia. Como la programación es idéntica para los tres sensores inerciales y, por su parte, igual para ambos láseres, sólo se presenta el código de la imu1 y del láser1 por simplificación, exclusivamente cambia la nomenclatura tanto para imuX, como para laserX.

Para el filtrado de los cinco sensores lo primero que se hace es buscar valores que se encuentren por encima del *valor_excep_time* en la columna de tiempos, la cual es la primera en los cinco ficheros procedentes de los sensores, haciendo uso de la función de Matlab *find,* la cual devuelve los índices donde se encuentran dichos valores en ese vector de tiempos, *i_imuX* o *i_laserX.* Con estos índices localizados, se procede a eliminar dicha línea dentro de todos los datos. Adicionalmente, en el láser se hace un borrado adicional si el índice indicado es menos de 4, ya que se ha comprobado que el láser presenta valores espúreos en la toma de sus tres valores, proporcionando tiempos muy elevados, debido al tiempo de sincronización, por ello no son datos relevantes al ser dentro de los primeros microsegundos de su ejecución.

Como añadido de mejora, se ha realizado una comprobación de la consecución secuencial en la toma de datos de los ficheros, ya que, con muy baja probabilidad (aproximadamente 10^{-4}), se presentaban valores no secuenciales, así que se opta por eliminar dichos valores al no ser relevantes y fruto de una pequeña discrepancia en la emisión de ese paquete por parte del sensor.

```
%% Filtrados según el tiempo

valor_excep_time = time_stop + 0.05;

% Filtrado fichero IMU 1
i_imu1 = find(imu1(:,1)*10^-6 > valor_excep_time);

for i = 1:length(i_imu1)
    imu1(i_imu1(i),:) = [];
    i_imu1 = i_imu1-1;
end

% Puede ocurrir que algunas tomas de datos de los ficheros tengan
un valor de la toma de tiempo que no corresponda consecutivamente
a las anteriores pero no sea un valor disparatado, por lo tanto se
revisa que todo el fichero sea secuencial en la toma de datos
k=1;
while k < length(imu1)
    if (imu1(k,1) > imu1(k+1,1))
        imu1(k,:) = [];
    end
    k = k+1;
end

 …
```

```
% Filtrado fichero LASER 1
i_laser1 = find(laser1(:,1)*10^-6 > valor_excep_time);

for i = 1:length(i_laser1)
    if (i_laser1(i) < 4)
        laser1(i_laser1(i),:) = [];
        i_laser1 = i_laser1-1;
    else
        laser1(i_laser1(i),:) = [];
        laser1(i_laser1(i)+1,:) = [];
        laser1(i_laser1(i)+2,:) = [];
        laser1(i_laser1(i)+3,:) = [];
        i_laser1 = i_laser1-4;
    end
end

% Se revisa que todo el fichero sea secuencial en la toma de datos
k=1;
while k < length(laser1)
    if (laser1(k,1) > laser1(k+1,1))
         laser1(k,:) = [];
    end
    k = k+1;
end
```

Una vez analizado el filtrado en función del tiempo, se procede a analizar la definición del filtro de valores espúreos obtenidos en la lectura del sensor. Al igual que con el filtrado anterior, se presenta la programación para una de las IMUs y uno de los láseres al ser igual la programación para los otros. Internamente, el procesamiento tanto para el láser como de la IMU para encontrar los valores excepcionales y el borrado de estos es el mismo, con la salvedad de que el *valor_excepcional* considerado para el láser se ha multiplicado por 100, ya que se ha considerado que 20 no es suficientemente elevado (pero si lo es para los valores que ofrecen la IMU), ya que el láser devuelve sus unidades de medida en micras y la IMU en milésimas. Adicionalmente, cabe decir que se ha añadido un *for k=2:7* para la IMU para que recorra desde la columna 2 hasta la 7 de su fichero, que es donde contiene los seis datos que ofrece. El láser presenta sus datos de medida en su segunda columna, por lo tanto, no es necesario añadir este for.

```
%% Filtrados según el valor de salida

valor_excepcional = 20;

% Filtrado fichero IMU 1
for k = 2:7
    i_imu1 = find(abs(imu1(:,k)) > valor_excepcional);

    for i = 1:length(i_imu1)
        imu1(i_imu1(i),:) = [];
        i_imu1 = i_imu1-1;
    end
end
…
```

```
% Filtrado fichero LASER 1
i_laser1 = find(abs(laser1(:,2))> valor_excepcional*100);

for i = 1:length(i_laser1)
    laser1(i_laser1(i),:) = [];
    i_laser1 = i_laser1-1;
end
…
```

Una vez se ha mostrado la definición de la función de filtrado de valores espúreos, tanto en tiempo como en valor de datos, se regresa a la ejecución del fichero en cuestión, *auto_full_representation_filtered.m,* tras la llamada a dicha función.

A continuación, con los valores de salida que presenta la función *peaks_filter,* se realiza de nuevo la misma función de adquisición del *time_stop*, para asegurar que este no ha presentado variación tras el filtrado, mostrado de nuevo el dispositivo en cuestión por la ventana de comandos de Matlab.

Tras ello, se procede a la representación gráfica de los datos de la misma forma que se ejecutaba en el fichero *auto_full_representation.m* cuya programación se ha explicado anteriormente en la sección 4.2 Automatización de post-procesamiento de datos de este documento.

5 POST-PROCESAMIENTO SEÑALES CÁMARAS

En este capítulo se presenta el tratamiento de los datos capturados por los sensores de auscultación de la vía, cuyo resultado es un pilar fundamental en este proyecto ya que, como objetivo global se tiene el estudio del impacto que ejerce la vía sobre el vehículo ferroviario. Como la respuesta dinámica del vehículo ha sido representada en el capítulo anterior, mediante el tratamiento de datos obtenidos de los sensores dispuestos para ello, este capítulo es complementario, ya que se presentan gráficas en movimiento en Matlab que muestran las perturbaciones del perfil del láser de proyección lineal sobre las vías en el desplazamiento del vehículo.

A continuación, se presentan dos ficheros programados en lenguaje M en Matlab para la representación comentada. Cabe decir que el uso de uno u otro radica en la existencia de datos recogidos por ambas cámaras de auscultación. Si ambos ficheros presentan datos, el proceso de la representación se ha automatizado y se hace de forma conjunta para ambas cámaras, sin necesidad de especificar los ficheros de entrada, al igual siguiendo una metodología similar a la presentada en el capítulo anterior.

5.1 Representación de datos de cada cámara de auscultación

La representación de los datos de los diferentes sensores se ha realizado, como se ha comentado anteriormente, utilizando el programa Matlab. Para poder realizar un análisis de las necesidades que presenta el programa Matlab para obtener como inputs los documentos de salida de las cámaras, es necesario un procesamiento de dichos ficheros al presentar una extensión propia **.sfer*. Dicha extensión fue creada por el personal de la empresa VirtualMech, cuyo trabajo está íntimamente ligado con el departamento en ciertas actividades, como el procesamiento de las imágenes de estas cámaras. Es por ello que el procesamiento de estos ficheros se hace con la ejecución de un archivo*.exe* ofrecido por ellos, el cual se denomina *BinPlainConverter.exe*. Los pasos a seguir para su ejecución se muestran en la guía del Anexo 3 de este documento en la sección *Procesamiento de Datos*, los cuales se detallan a continuación:

1. Copiar el ejecutable *BinPlainConverter.exe* y pegarlo en el directorio donde se encuentran los ficheros **.sfer* devueltos por una de las cámaras.
2. Abrir el ejecutable *BinPlainConverter.exe* y escribir el nombre completo con su extensión de uno de los ficheros **.sfer*. Generará un *.txt* con dicha información en el directorio actual. Se cerrará el ejecutable una vez realizada la operación, así que hay que abrirlo de nuevo para realizar la misma operación con el otro fichero.

Los ficheros generados en formato texto serán los procesados por las funciones de Matlab de esta sección, los cuales se pasan como parámetros de entrada de la función *video*:

- cam_left_file: fichero de salida procesado de cámara izquierda, (*_cam_left.sfer.txt). Cada fila consta de 600 columnas y está formada por valores comprendidos entre [0 400], al ser imágenes de 400x600 píxeles. Cada valor de cada foto representa el píxel situado en medio de la nube de píxeles de dicha columna. Por lo tanto, la foto del perfil que se desea ver estará formada por ceros al inicio y fin de dicha fila con valores distintos de ceros por el centro (en el caso de no haber ruido en la imagen)
- cam_right_file: fichero de salida procesado de cámara derecha, (*_cam_right.sfer.txt). Su contenido sigue la misma dinámica que el fichero anterior.

Se presenta el código de la función *video.m* a continuación, el cual puede encontrarse completo en el Anexo 2. Al inicio, se definen los parámetros característicos de los videos, como son alto y ancho (400x600) de los fotogramas, la frecuencia de muestreo (250 Hz) y se calcula con ella el tiempo de muestreo.

```
function video(cam_left_file,cam_right_file)

% Las imágenes son de 400x600 píxeles, se toman estos datos para
su representacion
alto = 400;
ancho = 600;

% La frecuencia de muestreo del vídeo es de 250 Hz
freq_m = 250;
t_m = 1/freq_m;
```

Se comprueba que los ficheros se han pasado como parámetro o no (indicando un 0 como argumento de la función). Una vez hecha la carga del fichero, se obtiene el número de fotogramas que tienen cada vídeo, tanto el obtenido por la cámara situada en el carril izquierdo como la cámara situada sobre el carril derecho, mediando la función *size*. Además, se crean los vectores (*ancho_foto_**) que se pasarán como parámetros a la función *plot* para representar el eje horizontal, por ello va desde 1 hasta el ancho de la foto, dando saltos de 1 en 1.

```
% Comprobación de existencia de ambos ficheros
if cam_left_file ~= 0
    % Cada fila es una imagen y cada columna la altura media (en
píxeles) de los bits en blanco. Tiene 600 columnas por ser la
imagen de 400x600 pixeles
    matriz_video_izq=load(cam_left_file);
    [num_fotos_izq,~] = size(matriz_video_izq);
    ancho_foto_izq = 1:ancho;
end

if cam_right_file ~= 0
    % Cada fila es una imagen y cada columna la altura media (en
píxeles) de
    % los bits en blanco. Tiene 600 columnas por ser la imagen de
400x600 pixeles
    matriz_video_der=load(cam_right_file);
    [num_fotos_der,~] = size(matriz_video_der);
    ancho_foto_der = 1:ancho;
end
```

Como se ha comentado, en este fichero se contempla la representación exclusiva de una sola cámara o de las dos si ambos ficheros contienen información, es por ello que se ha utilizado una concatenación de funciones *if* y *elseif* para su comprobación. Dentro de esta estructura, se ha empleado, primero se representa el primer fotograma dando estructura a la ventana gráfica, en el caso de que ambas cámaras hayan aportado datos, se utiliza *subplot* para representar ambas gráficas en una misma pantalla, uno a la izquierda y otro a la derecha como las posición relativa de las cámaras. Par continuar con la reproducción de los siguientes fotogramas en formato vídeo, se ha empleado bucle *for*, el cual permanece a la espera durante la frecuencia de muestreo *pause(freq_m)*, aportándole así la misma velocidad que el vídeo real capturado por las cámaras. Así mismo, se ha añadido las propiedades deseadas a las gráficas haciendo uso de la función *set*, la cual aporta parámetros configurables de las gráficas. Como es habitual, adicionalmente se ha añadido títulos denominando tanto a los ejes como a cada gráfica para una mejor comprensión de la persona que observe dichas gráficas. Cabe observar que se ha añadido la función *refresh*, la cual refresca la imagen evitando que las imágenes se solapen, quedando así una visualización de efecto video, de ahí el nombre del fichero.

```
if cam_left_file ~= 0
    if cam_right_file ~= 0
        figure;
        foto_izq = matriz_video_izq(1,:);
        foto_der = matriz_video_der(1,:);
```

```
    subplot(1,2,1)
    plot(ancho_foto_izq,foto_izq,'.');
    title('Cámara izquierda')
    set(gca,'yDir','reverse')
    axis([1 ancho 1 alto]);
    xlabel('píxeles'),ylabel('píxeles')

    subplot(1,2,2)
    plot(ancho_foto_der,foto_der,'.');
    title('Cámara derecha')
    set(gca,'yDir','reverse')
    axis([1 ancho 1 alto]);
    xlabel('píxeles'),ylabel('píxeles')



    for i = 2:num_fotos_izq
        foto_izq = matriz_video_izq(i,:);
        foto_der = matriz_video_der(i,:);

        pause(freq_m)
        subplot(1,2,1)
        plot(ancho_foto_izq,foto_izq,'.');
        set(gca,'yDir','reverse')
        axis([1 ancho 1 alto]);
        xlabel('píxeles'),ylabel('píxeles')
        title('Cámara izquierda')
        refresh(1)

        subplot(1,2,2)
        plot(ancho_foto_der,foto_der,'.');
        set(gca,'yDir','reverse')
        axis([1 ancho 1 alto]);
        xlabel('píxeles'),ylabel('píxeles')
        title('Cámara derecha')
        refresh(1)
    end



elseif cam_right_file == 0
    foto_izq = matriz_video_izq(1,:);

    plot(ancho_foto_izq,foto_izq,'.');
    title('Cámara izquierda')
    set(gca,'yDir','reverse')
    axis([1 ancho 1 alto]);
    xlabel('píxeles'),ylabel('píxeles')
```

```
        for i = 2:num_fotos_izq
            foto_izq = matriz_video_izq(i,:);

            pause(freq_m)
            plot(ancho_foto_izq,foto_izq,'.');
            set(gca,'yDir','reverse')
            axis([1 ancho 1 alto]);
            xlabel('píxeles'),ylabel('píxeles')
            title('Cámara izquierda')
            refresh(1)
        end
    end


elseif cam_left_file == 0
    foto_der = matriz_video_der(1,:);

    plot(ancho_foto_der,foto_der,'.');
    title('Cámara izquierda')
    set(gca,'yDir','reverse')
    axis([1 ancho 1 alto]);
    xlabel('píxeles'),ylabel('píxeles')

    for i = 2:num_fotos_der
        foto_der = matriz_video_der(i,:);

        pause(freq_m)
        plot(ancho_foto_der,foto_der,'.');
        set(gca,'yDir','reverse')
        axis([1 ancho 1 alto]);
        xlabel('píxeles'),ylabel('píxeles')
        title('Cámara izquierda')
        refresh(1)
    end
end

end
```

5.2 Automatización de representación de datos de cada cámara de auscultación

En el apartado anterior se muestra la función *video.m* creada para la representación de la curva que presenta el láser de proyección sobre el carril al ser grabado por las cámaras de auscultación, de forma individual para el carril derecho y el izquierdo.

Al igual que se ha realizado para los sensores de estudio de la dinámica vehicular, se procede en esta sección a la automatización de la ejecución del fichero *video.m*, el cual se denomina *auto_video.m*, el cual no precisa de ficheros de entrada, exclusivamente hay que indicarle el PATH del ensayo en cuestión, abriendo el fichero y pegando dicho PATH en la primera variable que aparece, denominada *directorio*. Es preciso que este precedido y finalice con el carácter ‘’ para que dicha variable sea una cadena de caracteres, es decir, una variable de tipo *string*.

La primera parte de la función con mismo nombre del fichero, auto_full_representation, es donde radica la gran diferencia respecto al fichero anterior, ya que es donde se procede a la carga de ficheros de datos en lugar de pasarlos como argumentos de la función. Para ello, se declara la variable directorio a la cual se le asigna el PATH donde se encuentran situados todos los ficheros. Como requisito de buenas prácticas, ha de existir una carpeta por ensayo, es decir, sólo un fichero por cada sensor en dicho directorio correspondiente a un ensayo en exclusiva.

Al igual que se hizo para la función de automatización para sensores, se hav¡ce uso de la función de Matlab strcat, la cual unifica las dos cadenas de caracteres que se le pasan como argumentos. En esta ocasión se han concatenado la cadena de caracteres del PATH pasado a la variable *directorio* y **.sfer.txt*, de este modo, al aplicar la función de Matlab dir a dicha cadena de caracteres, busca todos los documentos que se encuentran dentro de ese directorio con la extensión de fichero **.sfer.txt* y los pasa a la variable estructura *d*. Para obtener los nombres de dichos ficheros, se emplea la nomenclatura de las variables estructuras para acceder a una de sus propiedades, *{d.name}*. De este modo, se pasan los nombres de los dicheros al vector *names_files*.

```
function auto_video
close all

%% Load files

% Se asignan los archivos de salida de los distintos componentes
en formato .csv
directorio = 'ensayos_alamillo_28_9_17\3\';
d = dir(strcat(directorio,'*.sfer.txt'));
names_files = {d.name};

cam_left_file = strcat(directorio,char(names_files(1)));
cam_right_file = strcat(directorio,char(names_files(2)));
```

Tras esta forma de carga de los ficheros, se procede a representar las imágenes obtenidas por las cámaras en modo video del mismo modo que se ha hecho para la función *video*.

6 ENSAYOS EN CAMPO

En el presente capítulo, se muestra la forma en que se realizaron los ensayos realizados al vehículo ferroviario instrumentado sobre las vías del Parque del Alamillo en Sevilla por parte del personal del departamento de Ingeniería Mecánica y Fabricación de la ESI junto con la colaboración de personal de la empresa VirtualMech. El proceso requiere de una fase previa de puesta a punto del sistema de adquisición de datos para su correcta ejecución. De la misma forma, se presentan los pasos a seguir para la ejecución del propio ensayo. Una vez realizado el mismo, se muestra cómo se procede a ejecutar el post-procesamiento de los datos in-situ tras realizar el ensayo, mostrando los resultados para poder verificar la correcta ejecución en todas las tomas de datos por parte de los sensores y poder realizar modificaciones en campo para poder realizar los diferentes escenarios que se consideren oportunos.

Como es objeto de este proyecto, se ha realizado una guía para el lector pueda reproducir las diferentes fases necesarias en la realización de los ensayos de forma detallada, la cual se presenta en el anexo 3 de este documento.

6.1 Puesta a punto

Para la realización de dichos ensayos, es necesaria una previa puesta a punto de todo el sistema de adquisición de datos:

1. Antes de salir a realizar el ensayo, es propicio asegurar que todo el cableado está bien conectado, incluyendo la comprobación con un dispositivo de continuidad que todas las conexiones del sistema intermedio están correctamente mirando esquema de Anexo 4 del proyecto Implementación y análisis de un sistema de adquisición de datos para un vehículo ferroviario a escala de María Dolores Sancho.
2. Cargar las batería un día previo a la realización del ensayo, ya que tardan varias horas en cargarse y para que no presenten mucha descarga si pasan muchos días cargadas
3. Una vez se está en campo, se comprueba si todos los dispositivos presentan buena sujeción al tren, desde los equipos de telecomunicación con son el router y el PC de abordo, como los diferentes sensores y sistemas intermedios.
4. Se alimenta el tren mediante baterías.
5. Encender el ordenador de abordo mediante la pestaña lateral y la bombilla parpadeará entre ámbar y verde. Cuando se quede fija en verde significará que se ha encendido.
6. Ahora se procede a realizar la conexión en remoto desde el PC de control al PC de a bordo del tren.
    a. Se realiza la conexión a la red wifi con la ID: "esfucon", y como contraseña: "esfucon16".
    b. Abrimos la aplicación "TeamViewer" y se introduce como ID de asociado: "192.168.1.40", estará seleccionada la opción "Control remoto" y se clicka en "Conectar con asociado". Como contraseña: "esfucon16"
7. Una vez realizada la conexión, se muestra el escritorio del PC del tren y con pestañas, las cuales se han de cerrar. Al estar en remoto cabe la posibilidad que la resolución de la pantalla no sea la más apropiada, así que se modifica en "Ver" -> "Resolución de pantalla"
8. Para comprobar la alineación de las cámaras, abrimos el programa "xiViewer" y se selecciona una de las dos cámaras. Para visualizar ambas, abrir de nuevo "xiViewer" y seleccionar la otra cámara.
   En el caso que no se visualice alguna cámara reiniciar el PC del tren con los USBs de las cámaras desenchufados y conectarlos cuando el PC esté encendido de nuevo. Además, visualizar las cámaras una a una, no las dos a la vez. Se enfocan manualmente las cámaras y, si se desea ver exclusivamente el lasér oscureciendo el fondo, se le da al botón derecho del ratón. En la ventana se quita la selección de "AEAG" y se puede variar la barra de "Exposure".
9. Se cierra xiViewer, ya que no se puede emplear este programa a la vez que la adquisición de datos.

6.2 Adquisición de datos

En esta sección se muestran los pasos a seguir para poner en marcha el tren y comenzar así el ensayo y la adquisición de datos por parte de los sensores:

1. Abrimos el programa "Qt Creator". Se selecciona en "Recent Projects" "tep7280_daqsysmicro". En la nueva ventana se clicka en Run (botón en la parte inferior izquierda de la pantalla, verde con forma de punta de flecha) o bien a Crtl+R, apareciéndonos una nueva ventana.
2. La ventana mostrada controla los diferentes componentes encargados de la adquisición de datos y controla el funcionamiento del motor. Lo que se utiliza son los recuadros titulados:
    a. MOTOR DRIVE
    b. Microcontroller acquiring on Serial Port 1
    c. Microcontroller acquiring on Serial Port 2
    d. Sync cameras
    e. Right camera
    f. Left camera

    La activación de cada uno de ellos se explica en los siguientes pasos.
    a. MOTOR DRIVE:
        i. En la barra correspondiente a "s" se selecciona el ángulo de la pendiente con la que se desea que el motor adquiera paulatinamente velocidad hasta que llegue a la que se desea, está en %. Normalmente, "s = 25", ha de ser tal que las ruedas del tren no patinen al iniciar la marcha.
        ii. La velocidad se encuentra en %, y se escribe directamente en el recuadro "vmax". Se puede poner 100% o lo que se desee, 0% para que se quede parado el motor.
        iii. Ya no se modifica nada más de este recuadro, los botones de "run & save", "run" o "stop" no se utilizan. Sólo destacar que la información que se adquiera del encoder del motor se guardará en el fichero indicado: "*_enc.csv", donde * es un distintivo de cada fichero generado en función de la fecha y hora en la que se realice.
    b. Microcontroller acquiring on Serial Port 1:
        i. "Activar/desactivar": seleccionar para activar la adquisición de datos por parte de la Mbed que tiene conectada dos IMUs y un láser, todo ello al puerto serie 1 (SP1). Aparecerá CONNECTED o DISCONNECTED" en "Estado".
        ii. El puerto que suele reconocer el PC es COM25, (comprobar si es ese al desenchufar y enchufar la Mbed que contiene dos IMUs en "Inicio"->"Dispositivos e impresoras").
        iii. Los ficheros que devuelve es uno por cada IMU y otro perteneciente a la salida del láser de medición de distancias.
    c. Microcontroller acquiring on Serial Port 2:
        i. "Activar/desactivar": seleccionar para activar la adquisición de datos por parte de la Mbed que tiene conectada una IMU y un láser, todo ello al puerto serie 2 (SP2). Aparecerá CONNECTED o DISCONNECTED" en "Estado".
        ii. El puerto que suele reconocer el PC es COM24, (comprobar si es ese al desenchufar y enchufar la Mbed que contiene una imu en "Inicio"->"Dispositivos e impresoras").
        iii. Los ficheros que devuelve es uno por cada imu y otro perteneciente a la salida del láser de medición de distancias.

d. Sync cameras:
   i. "Activar/desactivar": seleccionar para activar la adquisición de datos por parte de la Mbed que envía un pulso para informar a las cámaras de que empiecen a grabar.
   ii. El puerto que suele reconocer el PC es COM14, (comprobar si es ese al desenchufar y enchufar la Mbed en "Inicio"->"Dispositivos e impresoras").
   iii. No devuelve fichero ya que no es relevante.
e. Right camera:
   i. "Activar/desactivar": seleccionar para activar la adquisición de datos por parte de la cámara. Aparecerá "CONNECTED" o "DISCONNECTED" en "Estado".
   ii. Se asigna automáticamente el número de serie de la cámara, "SN: 07431751".
   iii. El fichero que devuelve está compuesto por líneas representantes de las fotos del vídeo grabado por la cámra, las cuales tienen 600 columnas, al ser las fotos de 600x400 píxeles.
f. Left camera:
   i. "Activar/desactivar": seleccionar para activar la adquisición de datos por parte de la cámara. Aparecerá "CONNECTED" o "DISCONNECTED" en "Estado".
   ii. Se asigna automáticamente el número de serie de la cámara.
   iii. El fichero que devuelve está compuesto por líneas representantes de las fotos del vídeo grabado por la cámara, las cuales tienen 600 columnas, al ser las fotos de 600x400 píxeles.
g. Una vez activados los elementos que se deseen, se da al boton Start adquisition (botón azul con punta de flecha blanca, en la parte superior de la pantalla).
h. Cuando se desee parar la ejecución, se pulsa el botón Stop adquisition (botón rojo con cuadrado blanco, en la parte superior de la pantalla).

6.3 Post-procesamiento de datos

Una vez se han realizado los ensayos, se procede a la ejecución del post-procesado de los datos, descrito en el capítulo 4 de este documento, obtenidos por los sensores:

1. Todos los ficheros de las adquisiciones se guardan dentro del PC en el directorio: "C:\Users\Virtualmech\Documents\Tep 7280\tep7280_daqsysmicro-copia_results"
2. Si se desean visualizar lo datos de las cámaras de forma rápida, se abre el ejecutable "adqCam". En la ventana se escribe "3" y se presiona ENTER. Tras ello, se escribe el PATH incluyendo el nombre del fichero de salida de la cámara (es válido pegar con el botón derecho).
3. Para procesarlos, dependiendo de si se ha realizado la conexión directamente al PC del tren o bien por remoto:
   a. Directamente al PC del tren: se copian los ficheros deseados en un pen (que se conecta a un USB del PC) y se pasan a un ordenador que tenga instalado el programa Matlab.
   b. Remoto desde portátil: se copian los ficheros desde el TeamViewer y se pegan en nuestro ordenador en el directorio deseado.
4. Se ejecutará el programa creado para la representación de los distintos datos: "auto_full_representation", para ello es necesario previamente entrar en él y pegar el PATH de los ficheros en la variable "directorio".

5. Para la visualización en modo vídeo en Matlab de los dicheros devueltos de las cámaras, seguir los siguientes pasos:
   a. Copiar el ejecutable "BinPlainConverter.exe" y pegarlo en el directorio donde se encuentran los ficheros "*.sfer" devueltos por una de las cámaras.
   b. Abrir el ejecutable "BinPlainConverter.exe" y escribir el nombre completo con su extensión de uno de los ficheros "*.sfer". Generará un .txt con dicha información en el directorio actual. Se cerrará el ejecutable una vez realizada la operación, así que hay que abrirlo de nuevo para hacer lo mismo con el otro fichero.
6. Se ejecutará el programa creado para la representación en Matlab de los vídeos generados por las cámaras: "auto_video", para ello es necesario previamente entrar en él y pegar el PATH de los ficheros "*.sfer" en la variable "directorio".
   DATO: para distintas representaciones y procesamiento de los datos mirar las descripciones de los ficheros de Matlab para tal efecto.

6.4 Resultados

En esta última sección, se presentan los resultados de los diferentes ensayos realizados con el vehículo instrumentado el día 28 de septiembre de 2017 sobre las vías disponibles del Parque del Alamillo de Sevilla. Los resultados mostrados son los resultantes de la ejecución de los dos ficheros más representativos considerados por los analistas del departamento de Ingeniería Mecánica y Fabricación de la ESI. Uno de ellos es el fichero en Matlab *auto_full_representation.m*, el cual se describió en detalle en el capítulo 4 de este documento, que presenta los datos de los diferentes sensores encargados de capturar los datos correspondientes a la dinámica vehicular del tren. Del mismo modo, se presentan los resultados de la función *auto_full_representation_filtered.m,* para apreciar la diferencia entre los datos en bruto y los datos con eliminación de los valores espúreos. Finalmente, se presenta una imagen a modo de representación del resultado expuesto por el fichero de post-procesamiento de las señales de video adquiridos por las cámaras de auscultación de la vía, *auto_video.m*, el cual se explica en detalle en el capítulo 5 de este documento y pierde valor al no poder ser representado en este documento más que por una imagen, sin ser formato video.

La carpeta que contiene las carpetas con los diferentes ensayos es denominada *ensayos_alamillo_28_9_17* y se encuentra en el mismo directorio que los tres archivos de representación de la función. Por ello, a la variable *directorio* de las tres funciones de representación simplemente hay que asignarle el valor:

```
directorio = 'ensayos_alamillo_28_9_17\X\';
```

donde X es sustituido por el nombre de la carpeta del ensayo, los cuales de han denominado de forma numérica: ‘1’, ‘2’, ‘3’ y ‘4’,

6.4.1 Ensayo 1

Se presenta a continuación las gráficas correspondientes a los datos obtenidos por el enconder, que han sido representados en la primera gráfica y en junto con la velocidad filtrada en azul, para poder visualizar su media. En la segunda gráfica se puede observar el solape existente entre ambas curvas ya que presentan los mismos valores tanto el espacio calculado a partir de los datos reales como el espacio calculado con medición de intervalos de tiempos iguales. Dichas gráficas han sido representadas con la ejecución de la función *auto_full_representation.m.*

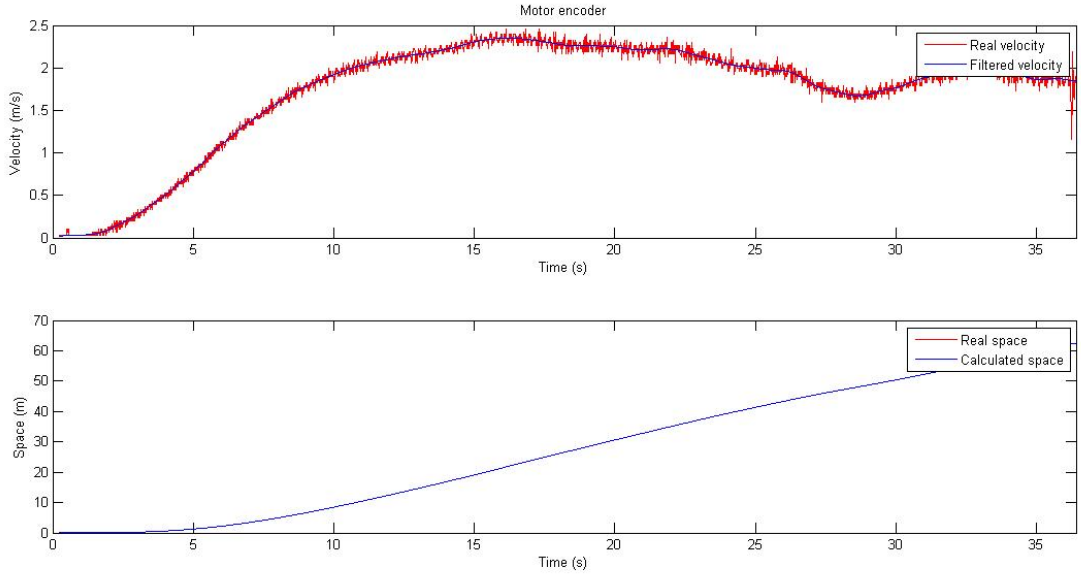


Figura 6-1. Ensayo 1. Representación datos brutos encoder.

A continuación, se muestran las seis gráficas correspondientes a los diferentes parámetros medidos por los sensores inerciales, que son las velocidades angulares en los tres ejes (x,y,z) y las aceleraciones en los tres ejes (x,y,z).

Se pueden visualizar perfectamente que en las diferentes gráficas hay unas líneas que no presentan concordancia con los valores medios, lo cual representan los datos espúreos comentados en el capítulo 4 de este documento, por ello, se muestra a continuación de dicha figura la representación de estas gráficas con tras haber realizado un filtrado de los datos esporádicos, gracias a la ejecución del mismo ensayo con la función *auto_full_represententation_filtered.m.*

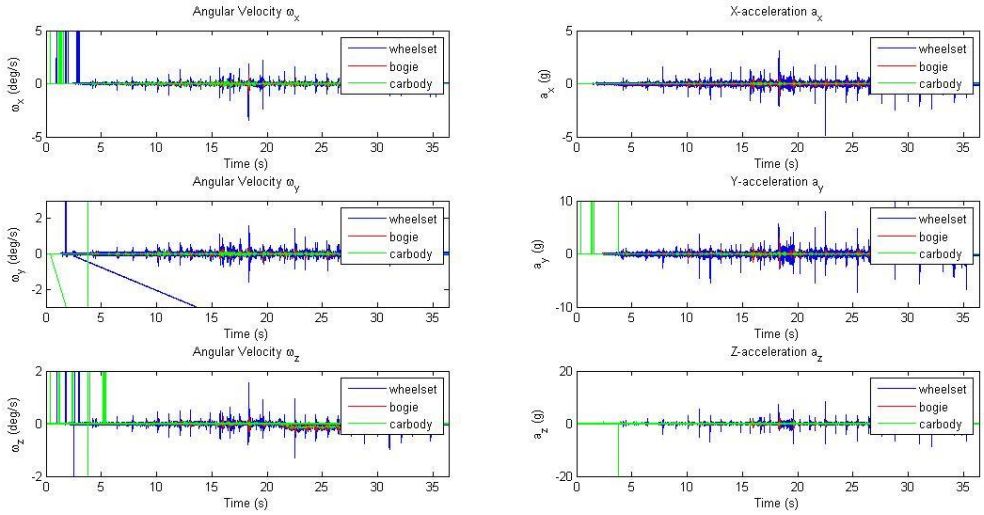


Figura 6-2. Ensayo 1. Representación datos brutos de IMUs.

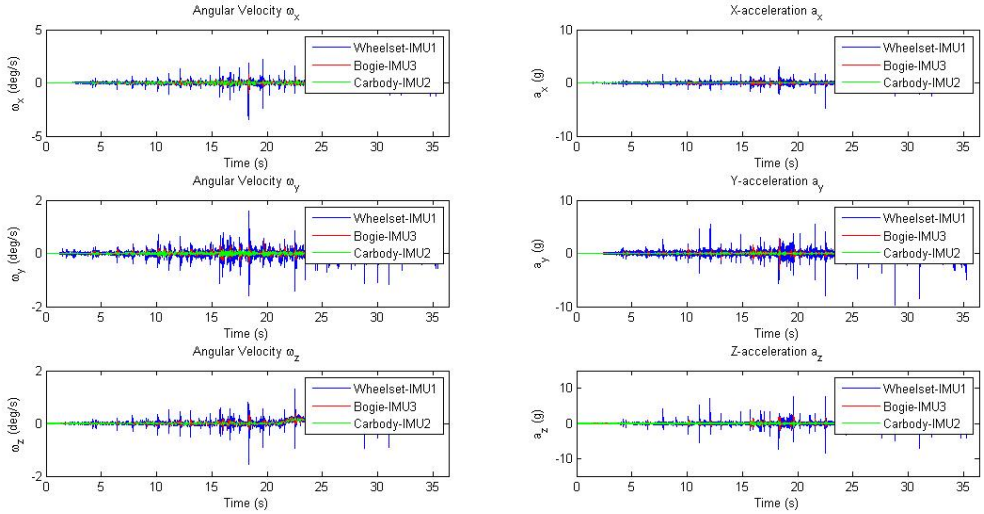


Figura 6-3. Ensayo 1. Representación datos filtrados de IMUs.

Del mismo, se ha realizado la representación de los datos obtenidos por los láseres de este ensayo, mostrándose en una primera imagen los datos sin filtrar y en la segunda filtrados. Ambas gráficas resultan idénticas en este ensayo, ya que los valores espúreos no aparecen en todos los ensayos. Se verá más adelante que los otros tres ensayos realizados sí aparecían estos picos de valores extremos.

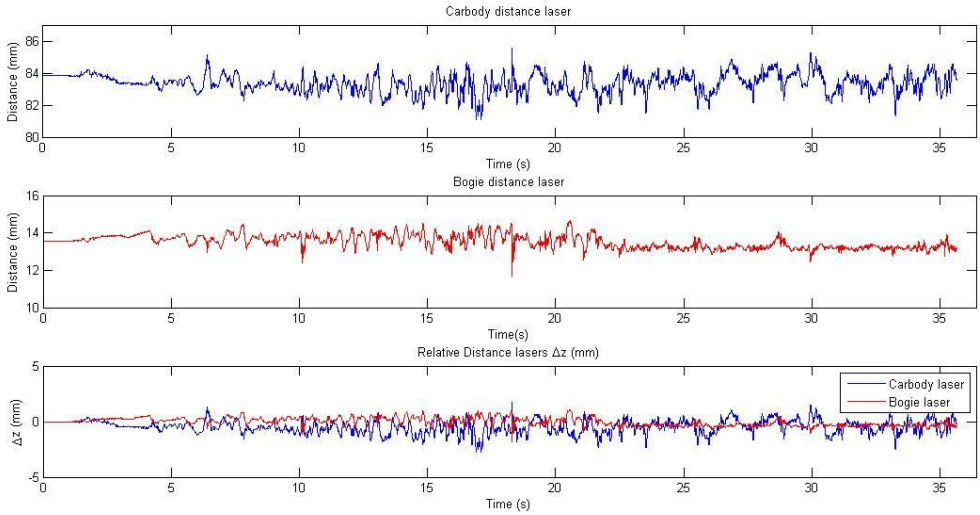


Figura 6-4. Ensayo 1. Representación datos brutos de láseres.

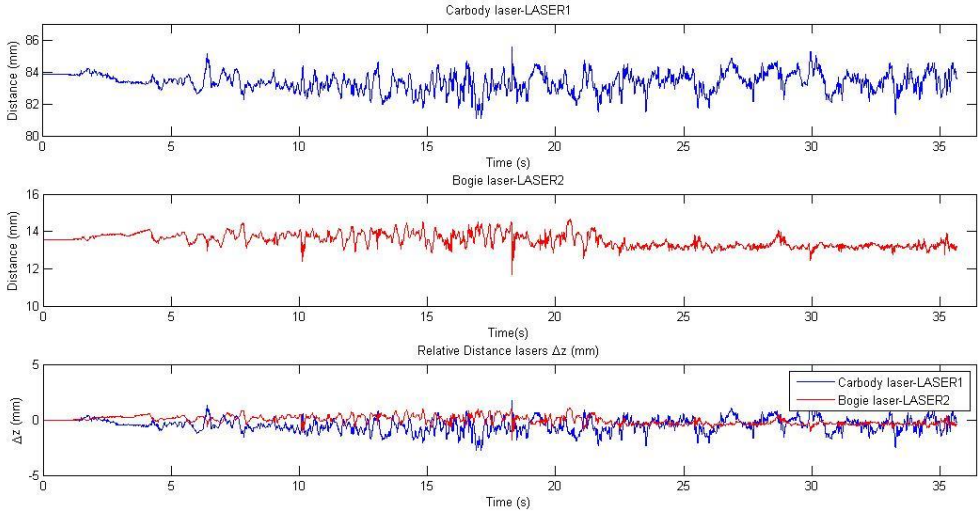


Figura 6-5. Ensayo 1. Representación datos filtrados de láseres.

6.4.2 Ensayo 2

Se presenta a continuación las gráficas correspondientes a los datos obtenidos por el enconder, que han sido representados en la primera gráfica y en junto con la velocidad filtrada en azul, para poder visualizar su media. En la segunda gráfica se puede observar el solape existente entre ambas curvas ya que presentan los mismos valores tanto el espacio calculado a partir de los datos reales como el espacio calculado con medición de intervalos de tiempos iguales. Dichas gráficas han sido representadas con la ejecución de la función *auto_full_representation.m.*

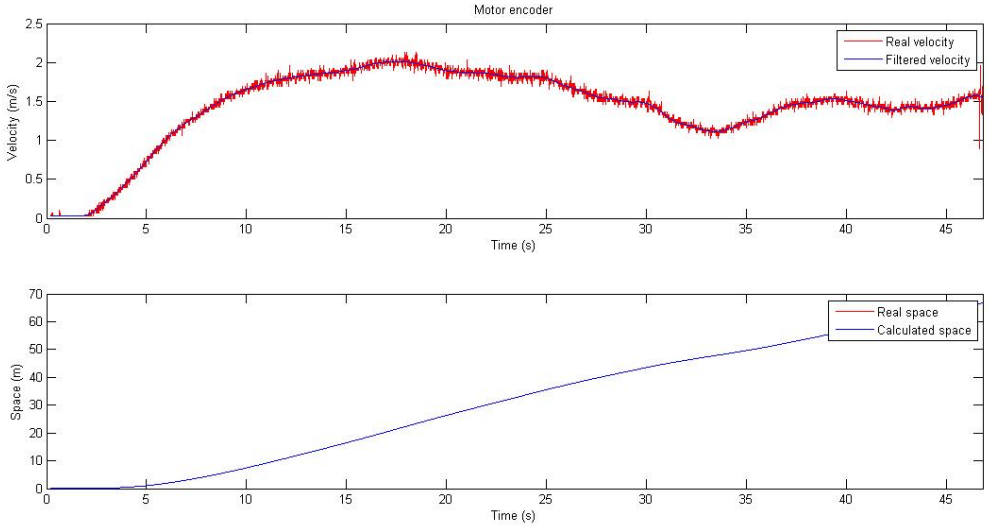


Figura 6-6. Ensayo 2. Representación datos brutos encoder.

A continuación, se muestran las seis gráficas correspondientes a los diferentes parámetros medidos por los sensores inerciales, que son las velocidades angulares en los tres ejes (x,y,z) y las aceleraciones en los tres ejes (x,y,z).

Se pueden visualizar perfectamente que en las diferentes gráficas hay unas líneas que no presentan concordancia con los valores medios, lo cual representan los datos espúreos comentados en el capítulo 4 de este documento, por ello, se muestra a continuación de dicha figura la representación de estas gráficas con tras haber realizado un filtrado de los datos esporádicos, gracias a la ejecución del mismo ensayo con la función *auto_full_represententation_filtered.m.*

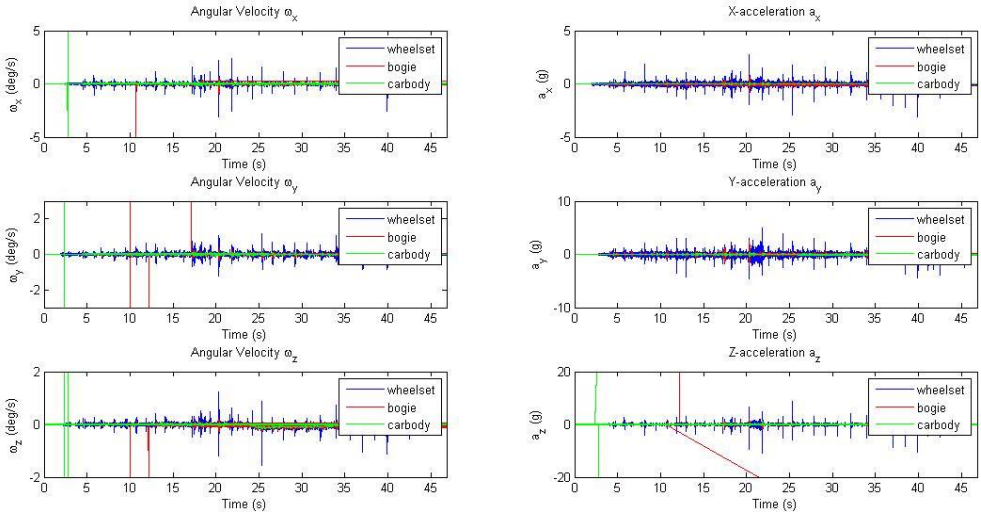


Figura 6-7. Ensayo 2. Representación datos brutos de IMUs.

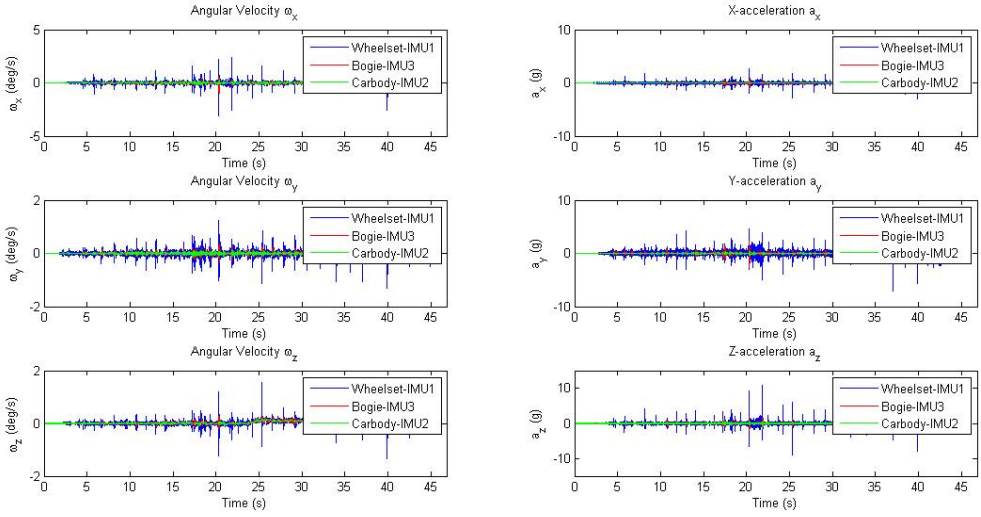


Figura 6-8. Ensayo 2. Representación datos filtrados de IMUs.

Del mismo, se ha realizado la representación de los datos obtenidos por los láseres de este ensayo, mostrándose en una primera imagen los datos sin filtrar, presentando datos espúreos que no son relevantes y perjudican a la representación de los otros datos al presentar valores muy desproporcionados.

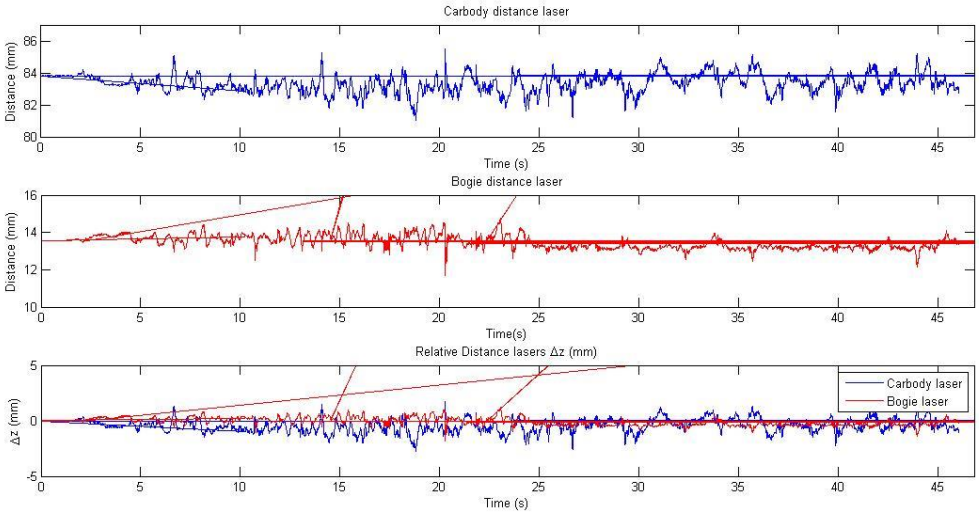


Figura 6-9. Ensayo 2. Representación datos brutos de láseres.

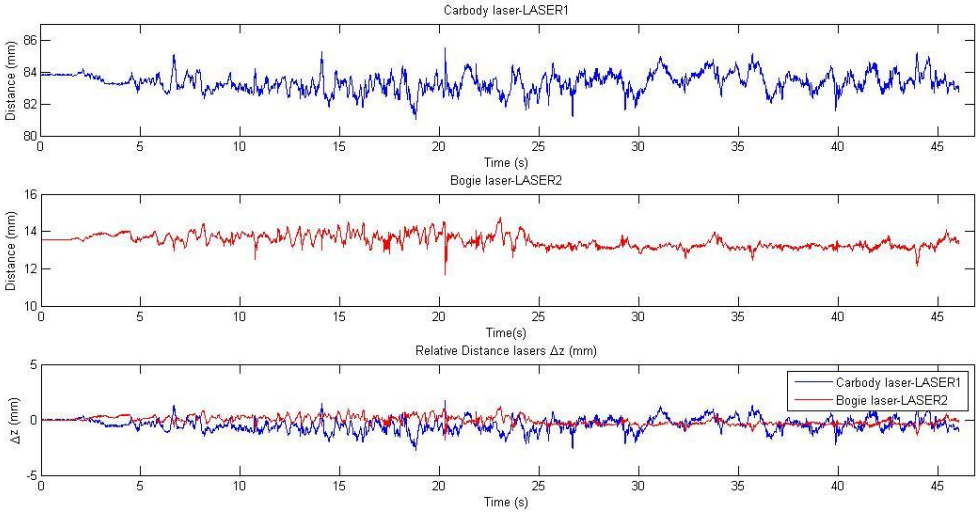


Figura 6-10. Ensayo 2. Representación datos filtrados de láseres.

6.4.3 Ensayo 3

Se presenta a continuación las gráficas correspondientes a los datos obtenidos por el enconder, que han sido representados en la primera gráfica y en junto con la velocidad filtrada en azul, para poder visualizar su media. En la segunda gráfica se puede observar el solape existente entre ambas curvas ya que presentan los mismos valores tanto el espacio calculado a partir de los datos reales como el espacio calculado con medición de intervalos de tiempos iguales. Dichas gráficas han sido representadas con la ejecución de la función *auto_full_representation.m.*

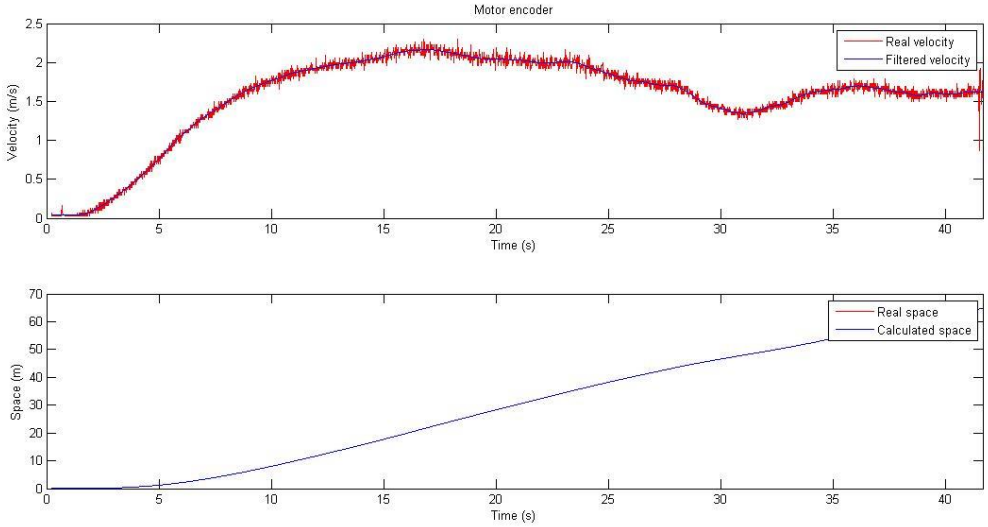


Figura 6-11. Ensayo 3. Representación datos brutos encoder.

A continuación, se muestran las seis gráficas correspondientes a los diferentes parámetros medidos por los sensores inerciales, que son las velocidades angulares en los tres ejes (x,y,z) y las aceleraciones en los tres ejes (x,y,z).

Se pueden visualizar perfectamente que en las diferentes gráficas hay unas líneas que no presentan concordancia con los valores medios, lo cual representan los datos espúreos comentados en el capítulo 4 de este documento, por ello, se muestra a continuación de dicha figura la representación de estas gráficas con tras haber realizado un filtrado de los datos esporádicos, gracias a la ejecución del mismo ensayo con la función *auto_full_represententation_filtered.m.*

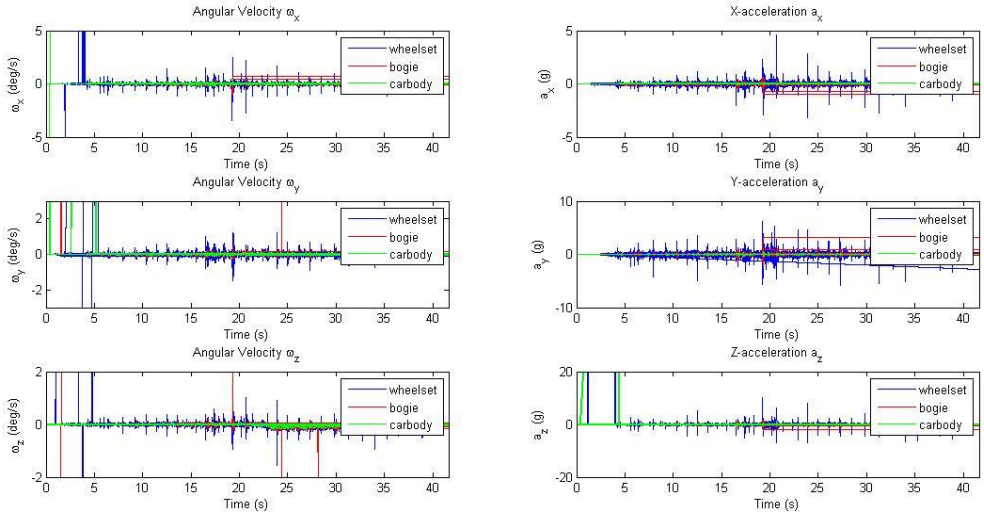


Figura 6-12. Ensayo 3. Representación datos brutos de IMUs.

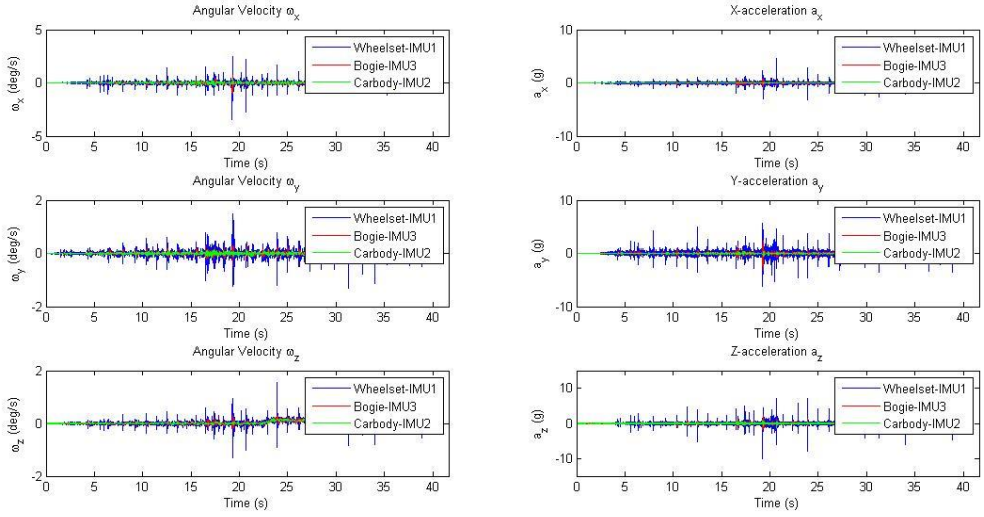


Figura 6-13. Ensayo 3. Representación datos filtrados de IMUs.

Del mismo, se ha realizado la representación de los datos obtenidos por los láseres de este ensayo, mostrándose en una primera imagen los datos sin filtrar, presentando datos espúreos que no son relevantes y perjudican a la representación de los otros datos al presentar valores muy desproporcionados.

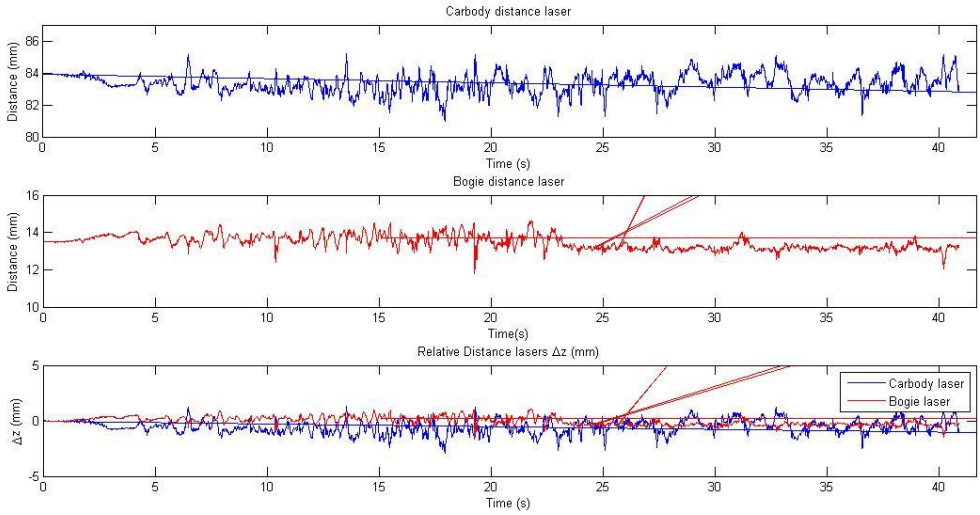


Figura 6-14. Ensayo 3. Representación datos brutos de láseres.

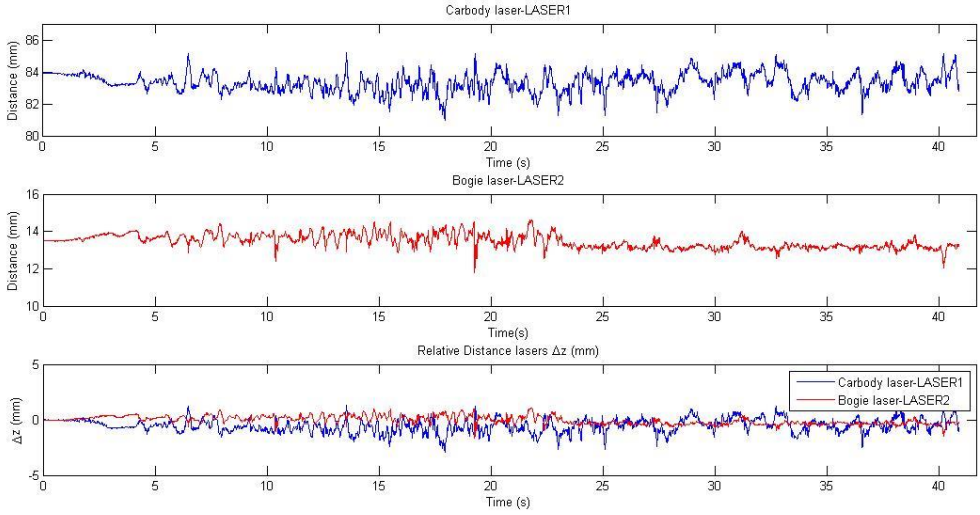


Figura 6-15. Ensayo 3. Representación datos filtrados de láseres.

6.4.4 Ensayo 4

Se presenta a continuación las gráficas correspondientes a los datos obtenidos por el enconder, que han sido representados en la primera gráfica y en junto con la velocidad filtrada en azul, para poder visualizar su media. En la segunda gráfica se puede observar el solape existente entre ambas curvas ya que presentan los mismos valores tanto el espacio calculado a partir de los datos reales como el espacio calculado con medición de intervalos de tiempos iguales. Dichas gráficas han sido representadas con la ejecución de la función *auto_full_representation.m.*

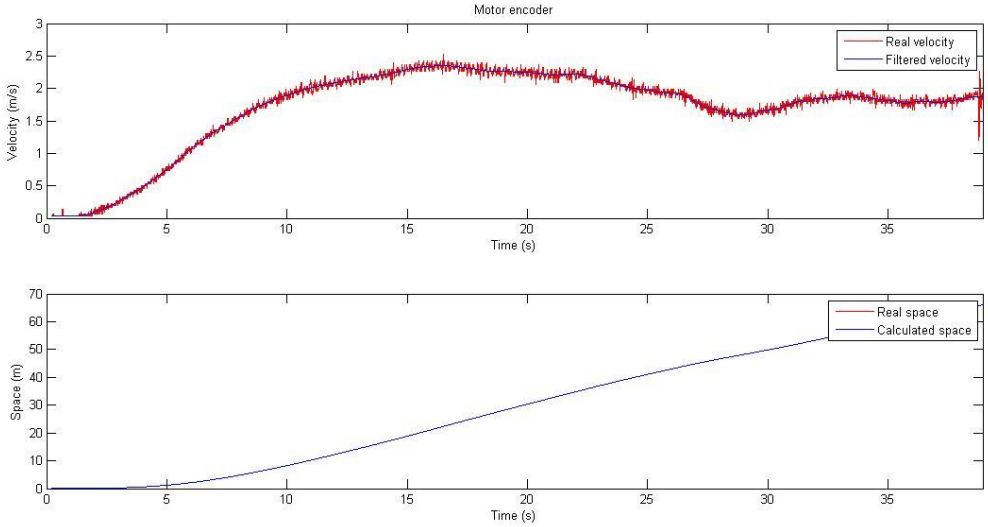


Figura 6-16. Ensayo 4. Representación datos brutos encoder.

A continuación, se muestran las seis gráficas correspondientes a los diferentes parámetros medidos por los sensores inerciales, que son las velocidades angulares en los tres ejes (x,y,z) y las aceleraciones en los tres ejes (x,y,z).

Se pueden visualizar perfectamente que en las diferentes gráficas hay unas líneas que no presentan concordancia con los valores medios, lo cual representan los datos espúreos comentados en el capítulo 4 de este documento, por ello, se muestra a continuación de dicha figura la representación de estas gráficas con tras haber realizado un filtrado de los datos esporádicos, gracias a la ejecución del mismo ensayo con la función *auto_full_represententation_filtered.m.*

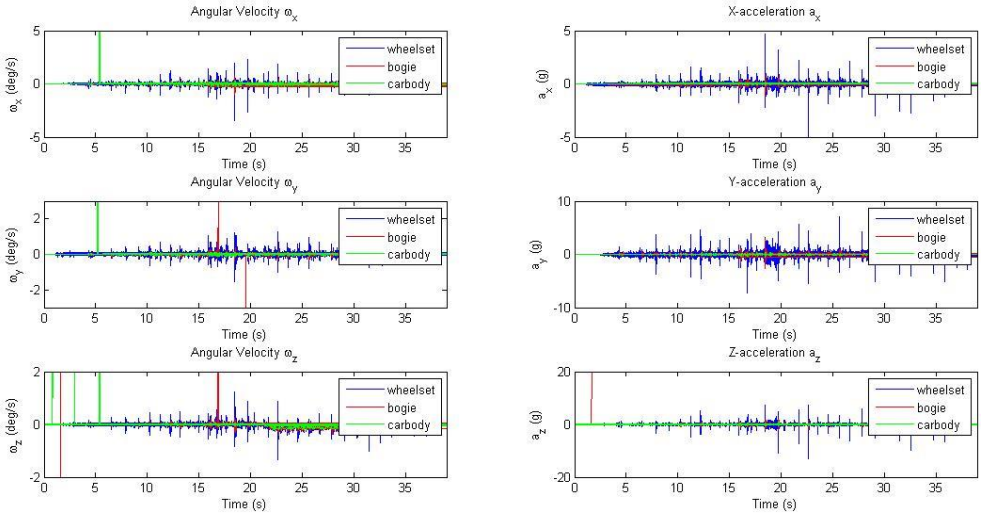


Figura 6-17. Ensayo 4. Representación datos brutos de IMUs.

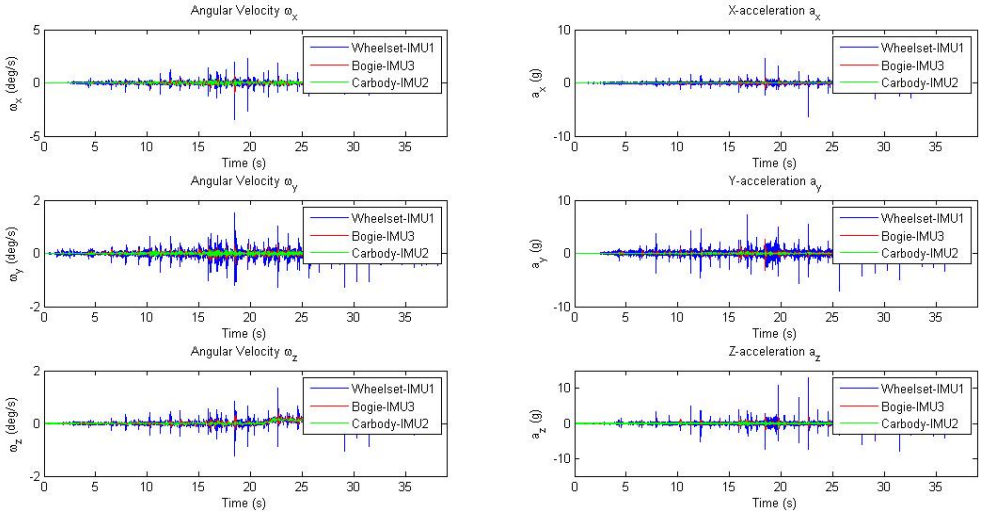


Figura 6-18. Ensayo 4. Representación datos filtrados de IMUs.

Del mismo, se ha realizado la representación de los datos obtenidos por los láseres de este ensayo, mostrándose en una primera imagen los datos sin filtrar, presentando datos espúreos que no son relevantes y perjudican a la representación de los otros datos al presentar valores muy desproporcionados.

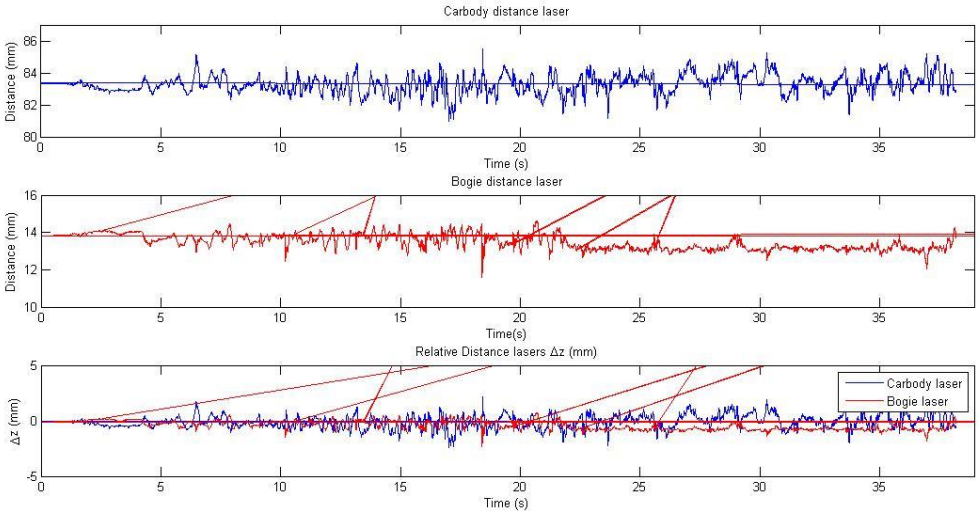


Figura 6-19. Ensayo 4. Representación datos brutos de láseres.

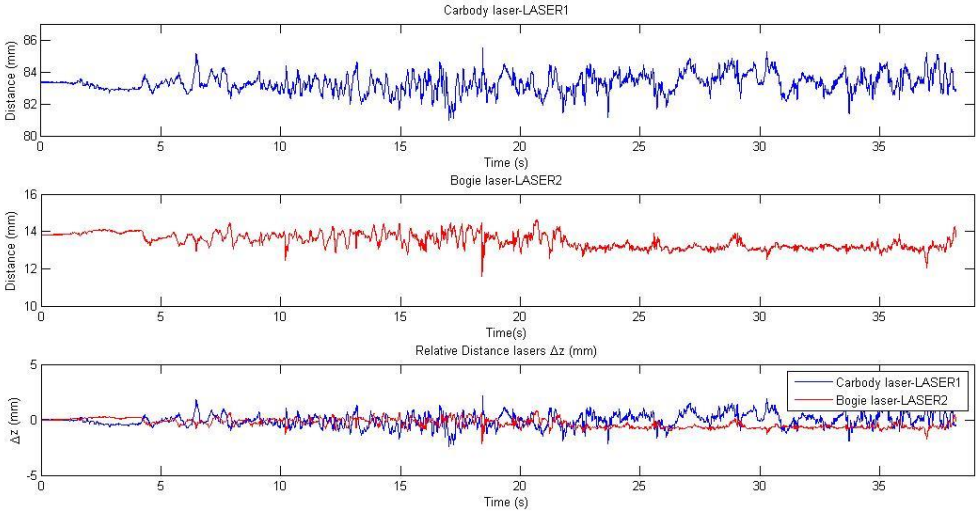


Figura 6-20. Ensayo 4. Representación datos filtrados de láseres.

Finalmente, se presenta una captura de pantalla como ejemplo de los resultados mostrados al ejecutar la función *auto_video* con los datos obtenidos en el ensayo 4. Como puede observarse, se presentan en ambas gráficas, tanto en la grabación correspondiente a la cámara izquierda como la cámara derecha, el perfil del carril izquierdo y derecho respectivamente, el cual es representado con una línea plana con una curvatura a la izquierda o derecha según corresponda. Gracias al análisis de los fotogramas de los vídeos grabados por las cámaras de auscultación y procesados para convertirlos en texto, se puede observar esta pequeña curva que se encuentra representada con en un nivel más elevado en ambas gráficas. Esto se corresponde con la curva generada del láser de proyección lineal sobre el carril. Este es uno de los principales objetivos de la realización de ensayos con el tren a instrumentado, la posibilidad de analizar las deformaciones que presenta el perfil de los carriles. Cabe comentar que la gráfica de la derecha presenta los valores más altos por el simple hecho de que la cámara está enfocado en ese plano más elevado a su correspondiente láser de proyección, el cual nos es indiferente mientras se pueda observar la variación del perfil del carril con la ejecución del vídeo. De este modo, se puede observar de mejor forma las desviaciones que presenta el carril en lugar de ser observados sobre un vídeo normal, ya que con los contrastes de colores y pequeñas reflexiones de la luz no se observaría tan definida la curva como aparece en estas gráficas. A modo de ejemplo de perturbación, se ha escogido esta captura de pantalla, ya que se puede observar en el fotograma obtenido por la cámara derecha unas deformaciones en el perfil bajo, representando la presencia de algún objeto sobre el que el láser está proyectando su perfil. Del mismo modo, si el carril presentara deformaciones, podría observarse fácilmente al igual que esta perturbación,

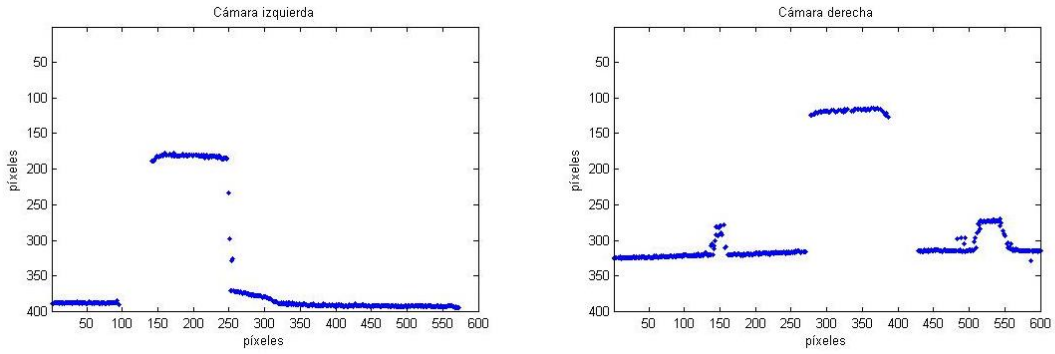


Figura 6-21. Ensayo 4. Captura de pantalla de reproducción de video de cámaras de auscultación.

7 CONCLUSIONES

Como cierre de este documento, se presentan en este capítulo las contribuciones generales que aporta el desarrollo de este proyecto al estudio en torno al vehículo ferroviario a escala que realiza el departamento de Ingeniería Mecánica y Fabricación, perteneciente a la Escuela Superior de Ingeniería de la Universidad de Sevilla.

El primer punto a destacar es el módulo principal presentado como post-procesamiento de datos capturados por los sensores, así como su tratamiento para la eliminación de los datos espúreos y la automatización del proceso de representación de datos. Este módulo capacita a los analistas a realizar el análisis de los datos obtenidos por los sensores de forma inmediata tras realizar un ensayo, comprobando si el ensayo ha transcurrido de forma correcta y se han obtenido todos los datos deseados por parte de los sensores, mostrando estos información relevante de las perturbaciones presentadas en el escenario que se quería estudiar.

El segundo punto es la descripción completa del sistema de adquisición de datos, el cual es fruto del trabajo realizado previamente a este por Juan Manuel Amador y tutorizado por José Luis Escalona [1]. De este modo se recoge en un único documento la información relevante referente a la estructura y funcionamiento de dicho sistema, así como se incluye el código para procesamiento de las señales, mapa de conexionado sistema intermedio y guía detallada para ejecutar ensayos.

Como último punto, quisiera comentar que todo este trabajo queda a disposición del lector para la posible continua mejora del sistema de adquisición de datos instalado en el vehículo ferroviario a escala objeto de estudio.

REFERENCIAS

[1] J. M. Amador Olivares y J. L. Escalona Franco, «Instrumentación de un vehículo ferroviario para la auscultación dinámica con sistemas de visión artificial,» Sevilla, 2016.

[2] The Qt company, «Qt,» 2019. [En línea]. Available: https://www.qt.io/.

[3] «LORD Microstrain GX4 Inertial Sensors,» Techni Measure, 2019. [En línea]. Available: https://www.technimeasure.co.uk/lord-microstrain-gx4-inertial-sensors/.

[4] «3DM-GX4 -25™ Attitude Heading And Reference System,» LORD sensing MicroStrain, 2019. [En línea]. Available: https://www.microstrain.com/inertial/3dm-gx4-25.

[5] MICRO-EPSILON, «Compact laser triangulation displacement sensor,» 2019. [En línea]. Available: https://www.micro-epsilon.com/displacement-position-sensors/laser-sensor/optoNCDT_1320/.

[6] MICRO-EPSILON, «Datasheet optoNCDT 1302,» 2019. [En línea]. Available: http://www.micro-epsilon.pl/download/man--optoncdt-1302--en.pdf.

[7] «MATLAB - Discover How to Solve Your Computational Problem,» MathWorks, 2019. [En línea]. Available: https://www.mathworks.com.

[8] «Filtro de Butterworth,» Wikipedia, 26 Octubre 2019. [En línea]. Available: https://es.wikipedia.org/wiki/Filtro_de_Butterworth.

1 ANEXOS: CÓDIGO C++ SISTEMA DE ADQUISICIÓN DE DATOS MBED MICROS

1.1 Programación microprocesador Mbed LPC11U24

```
// sistemaTacomero_Excelencia: main.cpp

#include "mbed.h"

// Parametros variables en cada aplicación*****************

#define FEQ_ACQ 250
// Se ha sustituido para fijar la frecuencia y obtener el periodo a
partir de esta.
//#define TSYNC 1250  // Periodo de sincronización en microsegundos

DigitalOut led1(p29);   // Salida digital conectada al led indicador
1
DigitalOut led2(p26);   // Salida digital conectada  al led
indicador 2

Serial pc(USBTX, USBRX);

DigitalIn sensor1(p19);     // Entrada digital del sensor 1
DigitalIn sensor2(p20);     // Entrada digital del sensor 1

Ticker leeSensor;           // Ticker con la función de leer
periódicamente la salida de los sensores magnéticos
Ticker sincro;              // Ticker con la función de cambiar de
estado la señal de sincronización

DigitalOut outS(p30);       // Señal de sincronización de las
cámaras (1)
DigitalOut outS2(p34);       // Señal de sincronización de las
cámaras (2)

DigitalOut myled(LED1);     // Led utilizado para debugging

Timer t;

int t_sync;

// Función que lee periódicamente la salida de los sensores
magnéticos
void lecturaSensores() {
    // Se envían los datos de los sensores por el puerto serie al PC
    /*pc.putc('1');       // Se envía un byte con el valor 1 antes
del valor del primer sensor
    pc.putc(sensor1);   // Se envía el valor digital actual del
sensor 1*/
    //pc.putc(rand()%2);  // Debugging
```

```
    led1 = sensor2;     // Se actualiza el estado del led indicador
1 de acuerdo al valor actual del sensor 1

    /*pc.putc('2');       // Se envía un byte con el valor 2 antes
del valor del segundo sensor
    pc.putc(sensor2);   // Se envía el valor digital actual del
sensor 1*/
    ///pc.putc(rand()%2);  // Debugging
    led2 = sensor1;     // Se actualiza el estado del led indicador
2 de acuerdo al valor actual del sensor 2
}

// Función que cambia de estado la señal de sincronización
void sincroniza() {
    outS = !outS;   // Se invierte el estado de la salida del pin
    outS2 = !outS2;
}

int main() {
    pc.baud(460800);    // Configuración de la velocidad de
comunicación del puerto serie
    char car;           // Variable auxiliar en la que se guardará
el valor del último caracter enviado por el PC
    myled = 0;  // Inicialización de señales

    // Prepara el periodo de sync de las camaras
    t_sync = 1000000/(2*FEQ_ACQ);
        pc.printf("Listo para enviar.\n\r");
        while(1) {  // Bucle infinito
        if(pc.readable()){  // Si existe algún byte en el buffer que
ha enviado el PC
            car = pc.getc();    // Se lee el byte
            if(car == 'a'){ // Si es el carácter 'a', indica el
inicio de la adquisición
                sincro.attach_us(&sincroniza, t_sync); // Se activa
el ticker de control de estado de la señal de sincronización con
periodo t_sync en microsegundos
                t.start();
                myled = 1;
            }else if(car == 's'){   // Si es el carácter 's', indica
el final de la adquisición
                // Se descativan los ticker
                sincro.detach();
                pc.printf("Duracion: %d\n\r",  t.read_us());
                t.stop();
                t.reset();
                myled = 0;  // Se resetean las señales
                led1 = 0;
                led2 = 0;
                outS = 1;   // La señal que sale del micro se va a
invertir al pasar por el transistor, por lo que para que la cámara
vea un flanco de subida al inicio de la captura de imágenes, se debe
enviar un flanco de subida desde el micro
                outS2 = 1;
            }
        }
    }
```

1.2 Programación microprocesador Mbed LPC1768 3 sensores

```
/**************************************
 VIRTUALMECH
 Autor: Juan Manuel Amador Olivares
 Fecha: 16/9/2015
 **************************************/


/**************************************
    Adquisición de datos IMU y Láser
 **************************************/
#include "mbed.h"
#include "Buffering.h"
#include "BufferBig.h"
#include "Bufferinguint.h"


#define TAMPAQUETEIMU 34    // Tamaño del paquete recibido de la IMU
#define TAMENVIOIMU 30       // Tamaño del paquete enviado al PC con los datos de la IMU
#define TAMENVIOLASER 8      // Tamaño del paquete enviado al PC con los datos del sensor de distancia


/********************
    Funciones
 ********************/
void startStreamingIMU();    // Manda los bytes a la IMU necesarios para que comience a enviar datos por streaming
void envioPaquete(unsigned char paquete[], int nElementos); // Envía por el puerto serie un paquete de datos
void envioDatos();           // Envio al PC los bytes de datos cuando es posible
void flushSerialBuffer1(void);
void flushSerialBuffer2(void);
void flushSerialBuffer3(void);
/********************/


DigitalOut led(LED1);   // Led indicador. Se enciende cuando se están adquiriendo datos
```

```
// Puertos UART a utilizar
Serial pLaser(p9, p10);
Serial pIMU(p28, p27);
Serial pIMU2(p13, p14);
Serial pc(USBTX, USBRX);


// Temporizador para el control del tiempo
Timer t;
unsigned int auxtime = 0;   // Variable auxiliar para la lectura del tiempo
unsigned char byteIN, byteINanterior, byteINbuff, byteINanteriorbuff; // Variables auxiliares para guardar bytes recibidos de la IMU
unsigned char byteIN2, byteINanterior2, byteINbuff2, byteINanteriorbuff2;    // Variables auxiliares para guardar bytes recibidos de la IMU2
unsigned char byteINL, byteINbuffL;  // Variable auxiliar para guardar bytes recibidos del laser


// Variables para la reconstrucción de los datos de la IMU
unsigned int nBytes = TAMPAQUETEIMU;
unsigned char paqueteEnvio[TAMENVIOIMU]; // 12 bytes para las aceleraciones (bytes cada eje) y otros 4 para la marca de tiempo y otros 12 para las velocidades angulares más un byte 'i' que indica que es el paquete de una IMU


// Variables para la reconstrucción de los datos de la IMU2
unsigned int nBytes2 = TAMPAQUETEIMU;
unsigned char paqueteEnvio2[TAMENVIOIMU]; // 12 bytes para las aceleraciones (bytes cada eje) y otros 4 para la marca de tiempo y otros 12 para las velocidades angulares más un byte 'i' que indica que es el paquete de una IMU2
/*union datoCompuesto{
    char B[4];
    float unidos;
} dato;*/


// Variables para la reconstrucción datos del láser
unsigned int distancia; // Sólo se usarán los 2 bytes menos significativos
unsigned char paqueteEnvioL[TAMENVIOLASER];
char fbyte = 0; // Indica si el byte leído es el primero delos dos enviados por el láser (tomando valor true) o el segundo (tomando valor
```

```
false)


// Buffers dónde se guardarán los bytes provenientes de los sensores
Buffering IMUbuff;
Bufferinguint IMUtime;
Buffering IMUbuff2;
Bufferinguint IMUtime2;
Buffering LASERbuff;
Bufferinguint LASERtime;


// Buffer dónde se guardarán los datos antes de ser enviados
BufferBig envioBuff;
char continuaEnviando;


// Variables semáforo para que no se mezclen los bytes de distintos
paquetes
// Estas variables indican de que sensor se pueden leer y procesar los
bytes de los paquetes recibidos.
// Independientemente de estas variables se recibirán los bytes de los
distintos sensores, que se irán guardando en sus respectivos buffer,
// y se irá guardando la marca de tiempo cuando se detecte la llegada
de un nuevo paquete.
bool enviandoIMU = false;
bool enviandoIMU2 = false;
bool enviandoLASER = false;
bool adquiriendo = false;


// Variable de control de bytes enviados
unsigned int bytesEnviados;
unsigned int selIMUleer;


int main() {
    // Inicialización de puertos UART
    pc.baud(460800);    // Configuración del puerto conectado al PC
    pIMU.baud(230400);  // Configuración del puerto conectado a la IMU
    pIMU2.baud(230400);  // Configuración del puerto conectado a la
IMU2
    pLaser.baud(115200);    // Configuración del puerto conectado al
sensor de distancia
```

```
    byteIN = 0;
    byteINbuff = 0;
    byteIN2 = 0;
    byteINbuff2 = 0;
    byteINL = 0;
    byteINbuffL = 0;

/*    t.reset();

    // Puertos serie
    flushSerialBuffer1();
    flushSerialBuffer2();
    flushSerialBuffer3();

    // Buffer en RAM de mbed
    while(!IMUbuff.isEmpty()){
        IMUbuff.get();
    }
    while(!IMUtime.isEmpty()){
        IMUtime.get();
    }
    while(!IMUbuff2.isEmpty()){
        IMUbuff2.get();
    }
    while(!IMUtime2.isEmpty()){
        IMUtime2.get();
    }
    while(!LASERbuff.isEmpty()){
        LASERbuff.get();
    }
    while(!LASERtime.isEmpty()){
        LASERtime.get();
    }
    while(!envioBuff.isEmpty()){
        envioBuff.get();
    }
```

```
  */
    //pc.printf("Listo para recibir datos.\n\r");

    while(1) {  // Bucle infinito
        // Se comprueba la llegada de nuevos bytes procedentes del PC
        if(pc.readable()){
            char c = pc.getc(); // Se lee el byte
            if(c == 'a'){   // Si es el carácter 'a', indica el inicio
de la adquisición
                // Puertos serie
                flushSerialBuffer1();
                flushSerialBuffer2();
                flushSerialBuffer3();

                // Buffer en RAM de mbed
                while(!IMUbuff.isEmpty()){
                    IMUbuff.get();
                }
                while(!IMUtime.isEmpty()){
                    IMUtime.get();
                }
                while(!IMUbuff2.isEmpty()){
                    IMUbuff2.get();
                }
                while(!IMUtime2.isEmpty()){
                    IMUtime2.get();
                }
                while(!LASERbuff.isEmpty()){
                    LASERbuff.get();
                }
                while(!LASERtime.isEmpty()){
                    LASERtime.get();
                }
                while(!envioBuff.isEmpty()){
                    envioBuff.get();
                }

                t.reset();
```

```
                t.start();

                //startStreamingIMU();
                bytesEnviados = 0;
                led = 1;
                adquiriendo = true;

            }else if(c == 's'){ // Si es el carácter 'a', indica el
final de la adquisición
                adquiriendo = false;
                led = 0;
                t.stop();
                //t.reset();

                // Puertos serie
                flushSerialBuffer1();
                flushSerialBuffer2();
                flushSerialBuffer3();

                // Buffer en RAM de mbed
                while(!IMUbuff.isEmpty()){
                    IMUbuff.get();
                }
                while(!IMUtime.isEmpty()){
                    IMUtime.get();
                }
                while(!IMUbuff2.isEmpty()){
                    IMUbuff2.get();
                }
                while(!IMUtime2.isEmpty()){
                    IMUtime2.get();
                }
                while(!LASERbuff.isEmpty()){
                    LASERbuff.get();
                }
                while(!LASERtime.isEmpty()){
                    LASERtime.get();
                }
```

```
                while(!envioBuff.isEmpty()){
                    envioBuff.get();
                }
                /*pc.printf("\n\nBytes       enviados:       %u\n\n",
bytesEnviados);
                pc.printf("\n\nDatos en el buffer de envio: %u, %u,
%u,           %u\n\n",           envioBuff.getDif(),IMUbuff.getDif(),
IMUbuff2.getDif(),LASERbuff.getDif());*/
            }
        }
        if(adquiriendo){     // Sólo se reconstruye y envían datos
cuando se ha iniciado la adquisición

            if(pIMU.readable()){
                byteINanterior = byteIN;
                byteIN = pIMU.getc();
                IMUbuff.put(byteIN);         // Se guarda el byte en
el buffer
                if(byteIN == 0x65 && byteINanterior == 0x75){// Si el
byte recibido es un 75 en hex, este byte marca la llegada de una nueva
medida
                    // Se guarda la marca de tiempo
                    IMUtime.put(t.read_us());
                }
            }

            // Se reciben los bytes de la IMU2, se guardan en un buffer
y se guarda una marca de tiempo si corresponde
            if(pIMU2.readable()){
                byteINanterior2 = byteIN2;
                byteIN2 = pIMU2.getc();
                IMUbuff2.put(byteIN2);        // Se guarda el byte en
el buffer
                if(byteIN2 == 0x65 && byteINanterior2 == 0x75){// Si
el byte recibido es un 75 en hex, este byte marca la llegada de una
nueva medida
                    // Se guarda la marca de tiempo
                    IMUtime2.put(t.read_us());
                }
            }

            // Se reciben los bytes del LASER, se guardan en un buffer
```

```
y se guarda una marca de tiempo si corresponde
            if(pLaser.readable()){
                byteINL = pLaser.getc();
                LASERbuff.put(byteINL); // Se guarda el byte en el
buffer
                if(byteINL > 127){     // Los bytes con el bit más
significativo a 1 son los primeros bytes d elos paquetes
                    // por lo que se guarda la marca de tiempo
correspondiente
                    LASERtime.put(t.read_us());
                }
            }


            if(!IMUbuff.isEmpty()  &&  enviandoIMU2  ==  false  &&
enviandoLASER == false){  // Si no se está enviando un paquete del
láser y existen bytes por leer
                byteINanteriorbuff = byteINbuff;
                byteINbuff = IMUbuff.get();
                // Reconstrucción del dato
                if(byteINbuff == 0x65 && byteINanteriorbuff == 0x75
&& nBytes >= (TAMPAQUETEIMU-1)){     // Si el byte recibido es un 75
en hex, este byte marca la llegada de una nueva medida
                    // Se pone el contador de bytes de un paquete a 0
                    //pc.putc('I');   // Se envían los dos bytes que
indican el comienzo del paquete
                    //pc.putc('I');     //antes de enviar el primer
bytes de datos
                    envioBuff.put('I');
                    envioBuff.put('I');
                    nBytes = 1;
                    enviandoIMU = true;      // Se está enviando un
paquete de IMU


                }else if(nBytes >= 1){  // Todo lo que no sea 0x75 es
parte del resto del dato
                    // Se suma un byte mas
                    nBytes++;
                    // Para reconstruir el dato se mandan primero los
bytes más significativos (que también es el orden en el que llegan)
                    switch(nBytes){
                    case 6:
                        //pc.putc(byteINbuff);
```

```
                envioBuff.put(byteINbuff);
                break;
            case 7:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
                break;
            case 8:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
                break;
            case 9:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
                break;
            case 10:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
                break;
            case 11:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
                break;
            case 12:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
                break;
            case 13:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
                break;
            case 14:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
                break;
            case 15:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
```

```
                    break;
                case 16:
                    //pc.putc(byteINbuff);
                    envioBuff.put(byteINbuff);
                    break;
                case 17:
                    //pc.putc(byteINbuff);
                    envioBuff.put(byteINbuff);
                    break;
                case 20:
                    //pc.putc(byteINbuff);
                    envioBuff.put(byteINbuff);
                    break;
                case 21:
                    //pc.putc(byteINbuff);
                    envioBuff.put(byteINbuff);
                    break;
                case 22:
                    //pc.putc(byteINbuff);
                    envioBuff.put(byteINbuff);
                    break;
                case 23:
                    //pc.putc(byteINbuff);
                    envioBuff.put(byteINbuff);
                    break;
                case 24:
                    //pc.putc(byteINbuff);
                    envioBuff.put(byteINbuff);
                    break;
                case 25:
                    //pc.putc(byteINbuff);
                    envioBuff.put(byteINbuff);
                    break;
                case 26:
                    //pc.putc(byteINbuff);
                    envioBuff.put(byteINbuff);
                    break;
```

```
                    case 27:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        break;
                    case 28:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        break;
                    case 29:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        break;
                    case 30:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        break;
                    case 31:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);

                        auxtime = IMUtime.get();      // Se recupera
la marca de tiempo del paquete y se envía dividido en 4 bytes
                        paqueteEnvio[3] = auxtime;
                        auxtime >>= 8;
                        paqueteEnvio[2] = auxtime;
                        auxtime >>= 8;
                        paqueteEnvio[1] = auxtime;
                        auxtime >>= 8;
                        paqueteEnvio[0] = auxtime;
                        /*pc.putc(paqueteEnvio[0]);
                        pc.putc(paqueteEnvio[1]);
                        pc.putc(paqueteEnvio[2]);
                        pc.putc(paqueteEnvio[3]);*/
                        envioBuff.put(paqueteEnvio[0]);
                        envioBuff.put(paqueteEnvio[1]);
                        envioBuff.put(paqueteEnvio[2]);
                        envioBuff.put(paqueteEnvio[3]);
                        enviandoIMU = false;       // Se ha terminado
```

```
de enviar el paquete de la IMU
                        bytesEnviados += TAMENVIOIMU;
                        break;
                    }
                }
            }


            if(!IMUbuff2.isEmpty()  &&  enviandoIMU  ==  false  &&
enviandoLASER == false){  // Si no se está enviando un paquete del
láser y existen bytes por leer
                byteINanteriorbuff2 = byteINbuff2;
                byteINbuff2 = IMUbuff2.get();
                // Reconstrucción del dato
                if(byteINbuff2 == 0x65 && byteINanteriorbuff2 == 0x75
&& nBytes2 >= (TAMPAQUETEIMU-1)){    // Si el byte recibido es un 75
en hex, este byte marca la llegada de una nueva medida
                    // Se pone el contador de bytes de un paquete a 0
                    //pc.putc('I');   // Se envían los dos bytes que
indican el comienzo del paquete
                    //pc.putc('I');    //antes  de  enviar  el  primer
bytes de datos
                    envioBuff.put('H');
                    envioBuff.put('H');
                    nBytes2 = 1;
                    enviandoIMU2 = true;      // Se está enviando un
paquete de IMU


                }else if(nBytes2 >= 1){  // Todo lo que no sea 0x75
es parte del resto del dato
                    // Se suma un byte mas
                    nBytes2++;
                    // Para reconstruir el dato se mandan primero los
bytes más significativos (que también es el orden en el que llegan)
                    switch(nBytes2){
                    case 6:
                        //pc.putc(byteINbuff2);
                        envioBuff.put(byteINbuff2);
                        break;
                    case 7:
                        //pc.putc(byteINbuff2);
                        envioBuff.put(byteINbuff2);
```

```
                    break;
                case 8:
                    //pc.putc(byteINbuff2);
                    envioBuff.put(byteINbuff2);
                    break;
                case 9:
                    //pc.putc(byteINbuff2);
                    envioBuff.put(byteINbuff2);
                    break;
                case 10:
                    //pc.putc(byteINbuff2);
                    envioBuff.put(byteINbuff2);
                    break;
                case 11:
                    //pc.putc(byteINbuff2);
                    envioBuff.put(byteINbuff2);
                    break;
                case 12:
                    //pc.putc(byteINbuff2);
                    envioBuff.put(byteINbuff2);
                    break;
                case 13:
                    //pc.putc(byteINbuff2);
                    envioBuff.put(byteINbuff2);
                    break;
                case 14:
                    //pc.putc(byteINbuff2);
                    envioBuff.put(byteINbuff2);
                    break;
                case 15:
                    //pc.putc(byteINbuff2);
                    envioBuff.put(byteINbuff2);
                    break;
                case 16:
                    //pc.putc(byteINbuff2);
                    envioBuff.put(byteINbuff2);
                    break;
```

```
                    case 17:
                        //pc.putc(byteINbuff2);
                        envioBuff.put(byteINbuff2);
                        break;
                    case 20:
                        //pc.putc(byteINbuff2);
                        envioBuff.put(byteINbuff2);
                        break;
                    case 21:
                        //pc.putc(byteINbuff2);
                        envioBuff.put(byteINbuff2);
                        break;
                    case 22:
                        //pc.putc(byteINbuff2);
                        envioBuff.put(byteINbuff2);
                        break;
                    case 23:
                        //pc.putc(byteINbuff2);
                        envioBuff.put(byteINbuff2);
                        break;
                    case 24:
                        //pc.putc(byteINbuff2);
                        envioBuff.put(byteINbuff2);
                        break;
                    case 25:
                        //pc.putc(byteINbuff2);
                        envioBuff.put(byteINbuff2);
                        break;
                    case 26:
                        //pc.putc(byteINbuff2);
                        envioBuff.put(byteINbuff2);
                        break;
                    case 27:
                        //pc.putc(byteINbuff2);
                        envioBuff.put(byteINbuff2);
                        break;
                    case 28:
```

```
                        //pc.putc(byteINbuff2);
                        envioBuff.put(byteINbuff2);
                        break;
                    case 29:
                        //pc.putc(byteINbuff2);
                        envioBuff.put(byteINbuff2);
                        break;
                    case 30:
                        //pc.putc(byteINbuff2);
                        envioBuff.put(byteINbuff2);
                        break;
                    case 31:
                        //pc.putc(byteINbuff2);
                        envioBuff.put(byteINbuff2);

                        auxtime = IMUtime2.get();     // Se recupera
la marca de tiempo del paquete y se envía dividido en 4 bytes
                        paqueteEnvio2[3] = auxtime;
                        auxtime >>= 8;
                        paqueteEnvio2[2] = auxtime;
                        auxtime >>= 8;
                        paqueteEnvio2[1] = auxtime;
                        auxtime >>= 8;
                        paqueteEnvio2[0] = auxtime;
                        envioBuff.put(paqueteEnvio2[0]);
                        envioBuff.put(paqueteEnvio2[1]);
                        envioBuff.put(paqueteEnvio2[2]);
                        envioBuff.put(paqueteEnvio2[3]);
                        enviandoIMU2 = false;      // Se ha terminado
de enviar el paquete de la IMU
                        bytesEnviados += TAMENVIOIMU;
                        break;
                    }
                }
            }

            if(!LASERbuff.isEmpty()   &&   enviandoIMU   ==   false   &&
enviandoIMU2 == false){
```

```
                byteINbuffL = LASERbuff.get();
                // Si el byte recibido tiene el bit más significativo
a 1 es el byte más significativo
                if(byteINbuffL > 127){
                    fbyte = 1;
                    distancia = byteINbuffL;
                    distancia &=~0x80;    // El bit mas significativo
hay que ponerlo a 0
                    enviandoLASER = true;    // Se comienza a enviar
el paquete del láser
                }else if (fbyte == 1){    // Si se recibió un byte más
significativo puede reconstruirse el valor medido por el laser
                    auxtime = LASERtime.get();  // Se lee el tiempo
cuando en el instante de llegada del primer byte del paquete
                    fbyte = 0;
                    distancia <<= 7;
                    distancia  += byteINbuffL;       // Unidades de
ingenieria
                    //distancia = distancia*0.0498;       // (mm) La
conversión a mm se hace en el PC para no tener que enviar un tipo
float
                                                          //        que
guardara los decimales de la operación
                    // Los dos primeros bytes byte de los paquetes del
laser seran una 'L'
                    paqueteEnvioL[0] = 'L';
                    paqueteEnvioL[1] = 'L';
                    paqueteEnvioL[3] = distancia;
                    distancia >>= 8;
                    paqueteEnvioL[2] = distancia;
                    // Se añaden también los 4 bytes del tiempo y se
envía el paquete (siempre el byte más significativo se envía primero)
                    paqueteEnvioL[7] = auxtime;
                    auxtime >>= 8;
                    paqueteEnvioL[6] = auxtime;
                    auxtime >>= 8;
                    paqueteEnvioL[5] = auxtime;
                    auxtime >>= 8;
                    paqueteEnvioL[4] = auxtime;
                    envioPaquete(paqueteEnvioL, TAMENVIOLASER);
                    enviandoLASER = false;     // Se ha terminado de
enviar el paquete del láser
```

```
                    bytesEnviados += TAMENVIOLASER;
                }


            }
            envioDatos();
        }     // if(adquiriendo)
    }   // while(1)
}       // main


// Envia todos los datos que pueda hasta que ya no se puede escribir
en el puerto o no haya más datos que enviar
void envioDatos(){
    continuaEnviando = 1;
    while(continuaEnviando){
        if (!envioBuff.isEmpty()){
            if(pc.writeable()){
                pc.putc(envioBuff.get());
                /*envioBuff.get();
                pc.putc('2');*/
            }else{
                continuaEnviando = 0;
            }
        }else{
           continuaEnviando = 0;
        }
    }
}


void envioPaquete(unsigned char paquete[], int nElementos){ // Envía
por el puerto serie un paquete de datos
    for(int i = 0; i < nElementos; i++){
        //pc.putc(paquete[i]);
        envioBuff.put(paquete[i]);
    }
}


void startStreamingIMU(){
    // Se envia un paquete de datos a la IMU que indica el comienzo
```

```
de lectura de medidas
    pIMU.putc(0x75); // K
    pIMU.putc(0x65); // A
    pIMU.putc(0x0C); // alt + 12 ♀
    pIMU.putc(0x05); // alt + 5 ♣
    pIMU.putc(0x05); // alt + 5 ♣
    pIMU.putc(0x11); // alt + 11 ♂
    pIMU.putc(0x01); // alt + 1 ☺
    pIMU.putc(0x01); // alt + 1 ☺
    pIMU.putc(0x01); // alt + 1 ☺
    pIMU.putc(0x04); // alt + 4 ♦
    pIMU.putc(0x1A); // alt + 26 →

    pIMU2.putc(0x75); // K
    pIMU2.putc(0x65); // A
    pIMU2.putc(0x0C); // alt + 12 ♀
    pIMU2.putc(0x05); // alt + 5 ♣
    pIMU2.putc(0x05); // alt + 5 ♣
    pIMU2.putc(0x11); // alt + 11 ♂
    pIMU2.putc(0x01); // alt + 1 ☺
    pIMU2.putc(0x01); // alt + 1 ☺
    pIMU2.putc(0x01); // alt + 1 ☺
    pIMU2.putc(0x04); // alt + 4 ♦
    pIMU2.putc(0x1A); // alt + 26 →
    //pc.printf("Paquete de inicio enviado.\n\r");
}


void flushSerialBuffer1(void) { char char1 = 0; while
(pIMU.readable()) { char1 = pIMU.getc(); } return; }
void flushSerialBuffer2(void) { char char1 = 0; while
(pIMU2.readable()) { char1 = pIMU2.getc(); } return; }
void flushSerialBuffer3(void) { char char1 = 0; while
(pLaser.readable()) { char1 = pLaser.getc(); } return; }
```

1.3 Programación microprocesador Mbed LPC1768 2 sensores

```
/**************************************
 VIRTUALMECH
 Autor: Juan Manuel Amador Olivares
 Fecha: 16/9/2015
 **************************************/

/**************************************
    Adquisición de datos IMU y Láser
 **************************************/
#include "mbed.h"
#include "Buffering.h"
#include "BufferBig.h"
#include "Bufferinguint.h"

#define TAMPAQUETEIMU 34    // Tamaño del paquete recibido de la IMU
#define TAMENVIOIMU 30       // Tamaño del paquete enviado al PC con los datos de la IMU
#define TAMENVIOLASER 8      // Tamaño del paquete enviado al PC con los datos del sensor de distancia

/********************
    Funciones
 ********************/
void startStreamingIMU();    // Manda los bytes a la IMU necesarios para que comience a enviar datos por streaming
void envioPaquete(unsigned char paquete[], int nElementos); // Envía por el puerto serie un paquete de datos
void envioDatos();           // Envio al PC los bytes de datos cuando es posible
void flushSerialBuffer1(void);
void flushSerialBuffer3(void);
/********************/

DigitalOut led(LED1);   // Led indicador. Se enciende cuando se están adquiriendo datos

// Puertos UART a utilizar
Serial pLaser(p9, p10);
```

```
Serial pIMU(p28, p27);
Serial pc(USBTX, USBRX);


// Temporizador para el control del tiempo
Timer t;
unsigned int auxtime = 0;   // Variable auxiliar para la lectura del tiempo
unsigned char byteIN, byteINanterior, byteINbuff, byteINanteriorbuff; // Variables auxiliares para guardar bytes recibidos de la IMU
unsigned char byteINL, byteINbuffL;  // Variable auxiliar para guardar bytes recibidos del laser


// Variables para la reconstrucción de los datos de la IMU
unsigned int nBytes = TAMPAQUETEIMU;
unsigned char paqueteEnvio[TAMENVIOIMU]; // 12 bytes para las aceleraciones (bytes cada eje) y otros 4 para la marca de tiempo y otros 12 para las velocidades angulares más un byte 'i' que indica que es el paquete de una IMU
/*union datoCompuesto{
    char B[4];
    float unidos;
} dato;*/


// Variables para la reconstrucción datos del láser
unsigned int distancia; // Sólo se usarán los 2 bytes menos significativos
unsigned char paqueteEnvioL[TAMENVIOLASER];
char fbyte = 0; // Indica si el byte leído es el primero delos dos enviados por el láser (tomando valor true) o el segundo (tomando valor false)


// Buffers dónde se guardarán los bytes provenientes de los sensores
Buffering IMUbuff;
Bufferinguint IMUtime;
Buffering LASERbuff;
Bufferinguint LASERtime;


// Buffer dónde se guardarán los datos antes de ser enviados
BufferBig envioBuff;
char continuaEnviando;
```

```
// Variables semáforo para que no se mezclen los bytes de distintos paquetes
// Estas variables indican de que sensor se pueden leer y procesar los bytes de los paquetes recibidos.
// Independientemente de estas variables se recibirán los bytes de los distintos sensores, que se irán guardando en sus respectivos buffer,
// y se irá guardando la marca de tiempo cuando se detecte la llegada de un nuevo paquete.
bool enviandoIMU = false;
bool enviandoLASER = false;
bool adquiriendo = false;

// Variable de control de bytes enviados
unsigned int bytesEnviados;

int main() {
    // Inicialización de puertos UART
    pc.baud(460800);    // Configuración del puerto conectado al PC
    pIMU.baud(460800);  // Configuración del puerto conectado a la IMU
    pLaser.baud(115200);    // Configuración del puerto conectado al sensor de distancia

    //pc.printf("Listo para recibir datos.\n\r");

    while(1) {  // Bucle infinito
        // Se comprueba la llegada de nuevos bytes procedentes del PC
        if(pc.readable()){
            char c = pc.getc(); // Se lee el byte
            if(c == 'a'){   // Si es el carácter 'a', indica el inicio de la adquisición
                adquiriendo = true;
                t.reset();
                t.start();
                startStreamingIMU();
                bytesEnviados = 0;
                led = 1;
            }else if(c == 's'){ // Si es el carácter 'a', indica el final de la adquisición
                adquiriendo = false;
                led = 0;
```

```
                t.stop();
                t.reset();
                // Se borra lo que quede en los buffer

                // Puertos serie
                flushSerialBuffer1();
                flushSerialBuffer3();

                // Buffer en RAM de mbed
                while(!IMUbuff.isEmpty()){
                    IMUbuff.get();
                }
                while(!IMUtime.isEmpty()){
                    IMUtime.get();
                }
                while(!LASERbuff.isEmpty()){
                    LASERbuff.get();
                }
                while(!LASERtime.isEmpty()){
                    LASERtime.get();
                }
                while(!envioBuff.isEmpty()){
                    envioBuff.get();
                }
                /*pc.printf("\n\nBytes       enviados:       %u\n\n",
bytesEnviados);
                pc.printf("\n\nDatos en el buffer de envio: %u\n\n",
envioBuff.getDif());*/
            }
        }

        if(adquiriendo){     // Sólo se reconstruye y envían datos
cuando se ha iniciado la adquisición

            // Se reciben los bytes de la IMU, se guardan en un buffer
y se guarda una marca de tiempo si corresponde
            if(pIMU.readable()){
                byteINanterior = byteIN;
                byteIN = pIMU.getc();
```

```
                IMUbuff.put(byteIN);         // Se guarda el byte en
el buffer
                if(byteIN == 0x65 && byteINanterior == 0x75){// Si el
byte recibido es un 75 en hex, este byte marca la llegada de una nueva
medida
                    // Se guarda la marca de tiempo
                    IMUtime.put(t.read_us());
                }
            }


            // Se reciben los bytes del LASER, se guardan en un buffer
y se guarda una marca de tiempo si corresponde
            if(pLaser.readable()){
                byteINL = pLaser.getc();
                LASERbuff.put(byteINL); // Se guarda el byte en el
buffer
                if(byteINL > 127){     // Los bytes con el bit más
significativo a 1 son los primeros bytes d elos paquetes
                    // por lo que se guarda la marca de tiempo
correspondiente
                    LASERtime.put(t.read_us());
                }
            }


            if(!IMUbuff.isEmpty() && enviandoLASER == false){  // Si
no se está enviando un paquete del láser y existen bytes por leer
                byteINanteriorbuff = byteINbuff;
                byteINbuff = IMUbuff.get();
                // Reconstrucción del dato
                if(byteINbuff == 0x65 && byteINanteriorbuff == 0x75
&& nBytes >= (TAMPAQUETEIMU-1)){     // Si el byte recibido es un 75
en hex, este byte marca la llegada de una nueva medida
                    // Se pone el contador de bytes de un paquete a 0
                    //pc.putc('I');   // Se envían los dos bytes que
indican el comienzo del paquete
                    //pc.putc('I');     //antes de enviar el primer
bytes de datos
                    envioBuff.put('I');
                    envioBuff.put('I');
                    nBytes = 1;
                    enviandoIMU = true;      // Se está enviando un
paquete de IMU
```

```
                }else{  // Todo lo que no sea 0x75 es parte del resto
del dato
                    // Se suma un byte mas
                    nBytes++;
                    // Para reconstruir el dato se mandan primero los
bytes más significativos (que también es el orden en el que llegan)
                    switch(nBytes){
                    case 6:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        break;
                    case 7:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        break;
                    case 8:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        break;
                    case 9:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        break;
                    case 10:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        break;
                    case 11:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        break;
                    case 12:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        break;
                    case 13:
                        //pc.putc(byteINbuff);
```

```
                envioBuff.put(byteINbuff);
                break;
            case 14:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
                break;
            case 15:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
                break;
            case 16:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
                break;
            case 17:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
                break;
            case 20:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
                break;
            case 21:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
                break;
            case 22:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
                break;
            case 23:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
                break;
            case 24:
                //pc.putc(byteINbuff);
                envioBuff.put(byteINbuff);
```

```
                        break;
                    case 25:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        break;
                    case 26:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        break;
                    case 27:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        break;
                    case 28:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        break;
                    case 29:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        break;
                    case 30:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        break;
                    case 31:
                        //pc.putc(byteINbuff);
                        envioBuff.put(byteINbuff);
                        auxtime = IMUtime.get();      // Se recupera
la marca de tiempo del paquete y se envía dividido en 4 bytes
                        paqueteEnvio[3] = auxtime;
                        auxtime >>= 8;
                        paqueteEnvio[2] = auxtime;
                        auxtime >>= 8;
                        paqueteEnvio[1] = auxtime;
                        auxtime >>= 8;
                        paqueteEnvio[0] = auxtime;
                        /*pc.putc(paqueteEnvio[0]);
```

```
                        pc.putc(paqueteEnvio[1]);
                        pc.putc(paqueteEnvio[2]);
                        pc.putc(paqueteEnvio[3]);*/
                        envioBuff.put(paqueteEnvio[0]);
                        envioBuff.put(paqueteEnvio[1]);
                        envioBuff.put(paqueteEnvio[2]);
                        envioBuff.put(paqueteEnvio[3]);
                        enviandoIMU = false;       // Se ha terminado
de enviar el paquete de la IMU
                        bytesEnviados += TAMENVIOIMU;
                        break;
                    }
                }
            }


            if(!LASERbuff.isEmpty() && enviandoIMU == false){
                byteINbuffL = LASERbuff.get();
                // Si el byte recibido tiene el bit más significativo
a 1 es el byte más significativo
                if(byteINbuffL > 127){
                    fbyte = 1;
                    distancia = byteINbuffL;
                    distancia &=~0x80;    // El bit más significativo
hay que ponerlo a 0. Para ello, se toma el valor en HEX 0x80 que en
DEC es 128 y en BIN 10000000.
                                          // Con ello, se hace una
operación AND bit a bit con el operador &= y el negado de 0x80,
quedando como:
                                          // distancia = distancia &
01111111
                    enviandoLASER = true;    // Se comienza a enviar
el paquete del láser
                }else{  // Recibido byte menos significativo
                    if (fbyte == 1){    // Si se recibió un byte más
significativo puede reconstruirse el valor medido por el laser
                        auxtime = LASERtime.get();   // Se lee el
tiempo cuando en el instante de llegada del primer byte del paquete
                        fbyte = 0;
                        distancia <<= 7;
                        distancia += byteINbuffL;     // Unidades de
ingenieria
                        //distancia = distancia*0.0498;      // (mm)
```

```
La conversión a mm se hace en el PC para no tener que enviar un tipo
float
                                                                //    que
guardara los decimales de la operación
                        // Los dos primeros bytes byte de los paquetes
del laser seran una 'L'
                        paqueteEnvioL[0] = 'L';
                        paqueteEnvioL[1] = 'L';
                        paqueteEnvioL[3] = distancia;
                        distancia >>= 8;
                        paqueteEnvioL[2] = distancia;
                        // Se añaden también los 4 bytes del tiempo y
se envía el paquete (siempre el byte más significativo se envía
primero)
                        paqueteEnvioL[7] = auxtime;
                        auxtime >>= 8;
                        paqueteEnvioL[6] = auxtime;
                        auxtime >>= 8;
                        paqueteEnvioL[5] = auxtime;
                        auxtime >>= 8;
                        paqueteEnvioL[4] = auxtime;
                        envioPaquete(paqueteEnvioL, TAMENVIOLASER);
                        enviandoLASER = false;    // Se ha terminado
de enviar el paquete del láser
                        bytesEnviados += TAMENVIOLASER;
                    }
                }


            }
            envioDatos();
        }     // if(adquiriendo)
    }   // while(1)
}       // main


// Envia todos los datos que pueda hasta que ya no se puede escribir
en el purto o no haya más datos que enviar
void envioDatos(){
    continuaEnviando = 1;
    while(continuaEnviando){
        if (!envioBuff.isEmpty()){
```

```
            if(pc.writeable()){
                pc.putc(envioBuff.get());
                //envioBuff.get();
                //pc.putc('2');
            }else{
                continuaEnviando = 0;
            }
        }else{
           continuaEnviando = 0;
        }
    }
}
void envioPaquete(unsigned char paquete[], int nElementos){ // Envía
por el puerto serie un paquete de datos
    for(int i = 0; i < nElementos; i++){
        //pc.putc(paquete[i]);
        envioBuff.put(paquete[i]);
    }
}
void startStreamingIMU(){
    // Se envia un paquete de datos a la IMU que indica el comienzo
de lectura de medidas
    pIMU.putc(0x75); // K
    pIMU.putc(0x65); // A
    pIMU.putc(0x0C); // alt + 12 ♀
    pIMU.putc(0x05); // alt + 5 ♣
    pIMU.putc(0x05); // alt + 5 ♣
    pIMU.putc(0x11); // alt + 11 ♂
    pIMU.putc(0x01); // alt + 1 ☺
    pIMU.putc(0x01); // alt + 1 ☺
    pIMU.putc(0x01); // alt + 1 ☺
    pIMU.putc(0x04); // alt + 4 ♦
    pIMU.putc(0x1A); // alt + 26 →
    //pc.printf("Paquete de inicio enviado.\n\r");
}
void   flushSerialBuffer1(void)   {   char   char1   =   0;   while
(pIMU.readable()) { char1 = pIMU.getc(); } return; }
void   flushSerialBuffer3(void)   {   char   char1   =   0;   while
(pLaser.readable()) { char1 = pLaser.getc(); } return; }
```

2 ANEXOS: CÓDIGO MATLAB POST-PROCESAMIENTO DE DATOS

2.1 Representación gráfica, con especificación de ficheros de entrada, de datos obtenidos mediante sensores

2.1.1 Representación datos sin filtrar obtenidos desde sensores – *full_representation.m*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%% DESCRIPCIÓN: fichero que representa los datos obtenidos
mediante el sistema de adquisición de tren a escala.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%% PARÁMETROS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
% encoder_file: fichero de salida encoder del motor, (*_enc.csv),
6 columnas:
%                time: representa el triple de la diferencia entre
dos muestras consecutivas de la columna "elapsed time (ms)" (NOTA:
no es utilizado para las representaciones de este fichero)
%                omega_m (rpm): velocidad del motor en revoluciones
por minuto (NOTA: si es negativo la polarización a la que se ha
conectado el moto es correcta)
%                encoder_position:
%                position_change:
%                elapsed time (ms): instante de tiempo de toma de
datos.
% imu1_file: fichero de salida IMU 1 conectada al puerto serie 1
(SP1, 2 IMUs conectadas a él), (*_sp1imu.csv), 7 columnas:
%                Time (us): instante de tiempo de toma de datos.
%                Angular rate X (°/s): velocidad angular obtenida
entorno al eje x, obtenida en grados/segundo
%                Angular rate Y (°/s): velocidad angular obtenida
entorno al eje y, obtenida en grados/segundo
%                Angular rate Z (°/s): velocidad angular obtenida
entorno al eje z, obtenida en grados/segundo
%                Acceleration X (g): aceleración obtenida entorno
al eje x, obtenida en g
%                Acceleration Y (g): aceleración obtenida entorno
al eje y, obtenida en g
%                Acceleration Z (g): aceleración obtenida entorno
al eje z, obtenida en g
% imu2_file: fichero de salida IMU 2 conectada al puerto serie 1
(SP1, 2 IMUs conectadas a él), (*_sp1imu2.csv). Mismas columnas
que imu1_file.
```

```
% imu3_file: fichero de salida IMU 3 conectada al puerto serie 2
(SP2, 1 IMU conectada a él), (*_sp2imu.csv). Mismas columnas que
imu1_file.
% laser1_file: fichero de salida laser 1 conectado al puerto serie
1 (SP1, 1 laser conectado a él), (*_sp1laser.csv), 2 columnas:
%               Time (us): instante de tiempo de toma de datos.
%               distance (mm): distancia desde la salida del laser
hasta donde se proyecta el punto del mismo (NOTA: para correctas
mediciones el led del laser ha de estar en verde)
% laser2_file: fichero de salida laser 2 conectado al puerto serie
2 (SP2, 1 laser conectado a él), (*_sp2laser.csv). Mismas columnas
que laser1_file.
% time_file: muestra el tiempo de ejecución del ensayo. Siempre
hay que ponerlo
%
% DATO: Las líneas de título de los ficheros ha de estar
comentadas para
% introducirlos como parámetros, es decir, han de estar precedidas
del
% símbolo "%" (sin comillas)
%
% DATO: Si un fichero no se desea representar, se escribe en su
lugar un cero "0"
% (sin comillas) como parámetro de la función
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

function
full_representation(encoder_file,imu1_file,imu2_file,imu3_file,las
er1_file,laser2_file,time_file)
close all

%% Time
% Se obtiene el tiempo de ejecución del ensayo para representar
las
% gráficas en dicho intervalo
fichero = fopen(time_file,'r');

% Se las dos primeras palabras de la cadena de caracteres
cadena_caracteres = fscanf(fichero,'%s',2);



% Se toma solo la parte que contiene el tiempo y se pasa a formato
numérico
% obteniendo el instante de tiempo en segundos donde se para la
ejecución
time_stop = str2num(cadena_caracteres(8:14));



%% Encoder

% Comprobación de representación del encoder.
% Carga del fichero de salida del mismo, pasado como parámetro de
esta
% función.
```

```
% El resultado se pasa a la función encargada de representar los
datos
% obtenidos, se puede consultar más abajo en este mismo fichero.
if encoder_file ~= 0
    encoder = load(encoder_file);
    figure
    hold on
    title('ENCODER')
    encoder_representacion(encoder,time_stop);
    hold off
end



%% IMUs

% Comprobación de representación de la IMU 1.
% Carga del fichero de salida de la misma, pasado como parámetro
de esta
% función.
% El resultado se pasa a la función encargada de representar los
datos
% obtenidos, se puede consultar más abajo en este mismo fichero.
if imu1_file ~= 0
    imu1 = load(imu1_file);
    figure
    hold on
    title('IMU 1')
    imu_representacion(imu1,time_stop);
    hold off
end

% Ídem para IMU 2.
if imu2_file ~= 0
    imu2 = load(imu2_file);
    figure
    hold on
    title('IMU 2')
    imu_representacion(imu2,time_stop);
    hold off
end

% Ídem para IMU 3.
if imu3_file ~= 0
    imu3 = load(imu3_file);
    figure
    hold on
    title('IMU 3')
    imu_representacion(imu3,time_stop);
    hold off
end



% Representación conjunta de las 3 IMUs
if imu1_file ~= 0
    if imu2_file ~= 0
```

```
        if imu3_file ~= 0
            % Angular velocities
            figure, plot(imu1(:,1)*10^-
6,imu1(:,2),'b',imu3(:,1)*10^-6,imu3(:,2),'r',imu2(:,1)*10^-
6,imu2(:,2),'g'), title('Angular Velocity \omega_x'), xlabel('Time
(s)'), ylabel('\omega_x (deg/s)'),
legend('wheelset','bogie','carbody'),xlim([0 time_stop])
            figure, plot(imu1(:,1)*10^-
6,imu1(:,3),'b',imu3(:,1)*10^-6,imu3(:,3),'r',imu2(:,1)*10^-
6,imu2(:,3),'g'), title('Angular Velocity \omega_y'), xlabel('Time
(s)'), ylabel('\omega_y (deg/s)'),
legend('wheelset','bogie','carbody'),xlim([0 time_stop])
            figure, plot(imu1(:,1)*10^-
6,imu1(:,4),'b',imu3(:,1)*10^-6,imu3(:,4),'r',imu2(:,1)*10^-
6,imu2(:,4),'g'), title('Angular Velocity \omega_z'), xlabel('Time
(s)'), ylabel('\omega_z (deg/s)'),
legend('wheelset','bogie','carbody'),xlim([0 time_stop])

            % Cartesian accelerations
            figure, plot(imu1(:,1)*10^-
6,imu1(:,5),'b',imu3(:,1)*10^-6,imu3(:,5),'r',imu2(:,1)*10^-
6,imu2(:,5),'g'), title('X-acceleration a_x'), xlabel('Time (s)'),
ylabel('a_x (g)'), legend('wheelset','bogie','carbody'),xlim([0
time_stop])
            figure, plot(imu1(:,1)*10^-
6,imu1(:,6),'b',imu3(:,1)*10^-6,imu3(:,6),'r',imu2(:,1)*10^-
6,imu2(:,6),'g'), title('Y-acceleration a_y'), xlabel('Time (s)'),
ylabel('a_y (g)'), legend('wheelset','bogie','carbody'),xlim([0
time_stop])
            figure, plot(imu1(:,1)*10^-
6,imu1(:,7),'b',imu3(:,1)*10^-6,imu3(:,7),'r',imu2(:,1)*10^-
6,imu2(:,7),'g'), title('Z-acceleration a_z'), xlabel('Time (s)'),
ylabel('a_z (g)'), legend('wheelset','bogie','carbody'),xlim([0
time_stop])

        %     % y-accel to compute curved stretch
        %     [B,A] = butter(4,0.05);
        %     figure, subplot(2,1,1),
plot(imu2(:,1),filtfilt(B,A,imu2(:,4)),'b'), xlabel('time (s)'),
ylabel('Carbody \omega_z filtered (rad/s)'), title('Transition
from tangent to curve stretch')
        %     subplot(2,1,2), plot(encoder(:,1),s2,'r'),
xlabel('time (s)'), ylabel('vehicle distance travelled (m)'),
title('Vehicle distance travelled')
        end
    end
end



%% Lasers

% Comprobación de representación de la laser 1.
% Carga del fichero de salida del mismo, pasado como parámetro de
esta
% función.
```

```
% El resultado se pasa a la función encargada de representar los
datos
% obtenidos, se puede consultar más abajo en este mismo fichero.
if laser1_file ~= 0
    laser1 = load(laser1_file);
    figure
    hold on
    title('LASER 1')
    laser_representacion(laser1,time_stop);
    hold off
end

% Ídem para laser 2.
if laser2_file ~= 0
    laser2 = load(laser2_file);
    figure
    hold on
    title('LASER 2')
    laser_representacion(laser2,time_stop);
    hold off
end

% Representación conjunta de los 2 lásers
% Comprobación de si se han pasado como parámetro los tres
ficheros correspondientes: "(imu1_file ~= 0)" devuelve un vector
fila de 1 de 1x27 en el caso de que se cumpla la condición.
% Se suman los valores de dichos vectores y ha de resultar 46 para
que se sepa que se han pasado los 2 ficheros y no un "0" en su
lugar como argumento.
if laser1_file ~= 0
    if laser2_file ~= 0
            % Distance lasers
            figure, subplot(3,1,1), plot(laser1(:,1)*10^-
6,laser1(:,2),'b'), xlabel('Time (s)'), ylabel('z (mm)'),
title('Carbody distance laser'),xlim([0 time_stop])
            subplot(3,1,2), plot(laser2(:,1)*10^-
6,laser2(:,2),'r'), xlabel('Time(s)'), ylabel('z (mm)'),
title('Bogie distance laser'),xlim([0 time_stop])
            subplot(3,1,3), plot(laser1(:,1)*10^-6,laser1(:,2)-
laser1(1,2),'b',laser2(:,1)*10^-6,laser2(:,2)-laser2(1,2),'r'),
title('Relative Distance lasers \Deltaz (mm)'), xlabel('Time
(s)'), ylabel('\Deltaz (mm)'), legend('Carbody laser','Bogie
laser'),,xlim([0 time_stop])
    end
end

fclose(fichero);

end


%% Funciones de representacion

function encoder_representacion(encoder,time_stop)
```

```
f_adq = 1/0.04;
f_nyq = f_adq/2;
f_c = 1;

[b,a] = butter(4,f_c/f_nyq,'low');
OMega = filtfilt(b,a,encoder(:,2));

plot(encoder(:,5)*10^-3,encoder(:,2),'b')
plot(encoder(:,5)*10^-3,OMega,'r')
xlabel('Time (s)'),ylabel('\omega_m (rpm)'),xlim([0 time_stop])

end



function imu_representacion(imu,time_stop)

subplot(3,2,1)
plot(imu(:,1)*10^-6,imu(:,2))
title('Angular rate (deg/s)')
xlabel('Time (s)'), ylabel('Angular rate X (deg/s)'),xlim([0
time_stop])
subplot(3,2,3)
plot(imu(:,1)*10^-6,imu(:,3))
xlabel('Time (s)'), ylabel('Angular rate Y (deg/s)'),xlim([0
time_stop])
subplot(3,2,5)
plot(imu(:,1)*10^-6,imu(:,4))
xlabel('Time (s)'), ylabel('Angular rate Z (deg/s)'),xlim([0
time_stop])
subplot(3,2,2)
plot(imu(:,1)*10^-6,imu(:,5))
title('Acceleration (g)')
xlabel('Time (s)'), ylabel('Acceleration X (g)'),xlim([0
time_stop])
subplot(3,2,4)
plot(imu(:,1)*10^-6,imu(:,6))
xlabel('Time (s)'), ylabel('Acceleration Y (g)'),xlim([0
time_stop])
subplot(3,2,6)
plot(imu(:,1)*10^-6,imu(:,7))
xlabel('Time (s)'), ylabel('Acceleration Z (g)'),xlim([0
time_stop])

end


function laser_representacion(laser,time_stop)

plot(laser(:,1)*10^-6,laser(:,2))
xlabel('Time (s)'), ylabel('Distance (mm)'),xlim([0 time_stop])


end
```

2.2 Representación gráfica automática de datos obtenidos mediante sensores

2.2.1 Representación automática de datos sin filtrar obtenidos desde sensores – *auto_full_representation.m*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%% DESCRIPCIÓN: fichero que representa los datos obtenidos de los
sensores
%               mediante el sistema de adquisición de datos de
tren a
%               escala.
%               Sólo es necesario pasar el directorio donde se
encuentran
%               todos los ficheros ".csv".
%               DATO: todos los ficheros han de contener
información, de lo
%               contrario utilizar la función
"full_representation.m"
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%% AUTOR: María Dolores Sancho Martín
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%% FICHEROS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
% encoder_file: fichero de salida encoder del motor, (*_enc.csv),
6 columnas:
%               time: representa el triple de la diferencia entre
dos muestras consecutivas de la columna "elapsed time (ms)" (NOTA:
no es utilizado para las representaciones de este fichero)
%               omega_m (rpm): velocidad del motor en revoluciones
por minuto (NOTA: si es negativo la polarización a la que se ha
conectado el moto es correcta)
%               encoder_position:
%               position_change:
%               elapsed time (ms): instante de tiempo de toma de
datos.
% imu1_file: fichero de salida IMU 1 conectada al puerto serie 1
(SP1, 2 IMUs conectadas a él), (*_sp1imu.csv), 7 columnas:
%               Time (us): instante de tiempo de toma de datos.
%               Angular rate X (°/s): velocidad angular obtenida
entorno al eje x, obtenida en grados/segundo
%               Angular rate Y (°/s): velocidad angular obtenida
entorno al eje y, obtenida en grados/segundo
%               Angular rate Z (°/s): velocidad angular obtenida
entorno al eje z, obtenida en grados/segundo
%               Acceleration X (g): aceleración obtenida entorno
al eje x, obtenida en g
%               Acceleration Y (g): aceleración obtenida entorno
al eje y, obtenida en g
```

```
%               Acceleration Z (g): aceleración obtenida entorno
al eje z, obtenida en g
% imu2_file: fichero de salida IMU 2 conectada al puerto serie 1
(SP1, 2 IMUs conectadas a él), (*_sp1imu2.csv). Mismas columnas
que imu1_file.
% imu3_file: fichero de salida IMU 3 conectada al puerto serie 2
(SP2, 1 IMU conectada a él), (*_sp2imu.csv). Mismas columnas que
imu1_file.
% laser1_file: fichero de salida laser 1 conectado al puerto serie
1 (SP1, 1 laser conectado a él), (*_sp1laser.csv), 2 columnas:
%               Time (us): instante de tiempo de toma de datos.
%               distance (mm): distancia desde la salida del laser
hasta donde se proyecta el punto del mismo (NOTA: para correctas
mediciones el led del laser ha de estar en verde)
% laser2_file: fichero de salida laser 2 conectado al puerto serie
2 (SP2, 1 laser conectado a él), (*_sp2laser.csv). Mismas columnas
que laser1_file.
%
% DATO: Las líneas de título de los ficheros ha de estar
comentadas para
% introducirlos como parámetros, es decir, han de estar precedidas
del
% símbolo "%" (sin comillas)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%

function auto_full_representation
close all


%% Load files

% Se asignan los archivos de salida de los distintos componentes
en formato .csv
directorio = 'ensayos_alamillo_28_9_17\2\';
d = dir(strcat(directorio,'*.csv'));
names_files = {d.name};

encoder_file = strcat(directorio,char(names_files(1)));
encoder = csvread(encoder_file,1,0);
imu1_file = strcat(directorio,char(names_files(2)));
imu1 = load(imu1_file);
imu2_file = strcat(directorio,char(names_files(3)));
imu2 = load(imu2_file);
laser1_file = strcat(directorio,char(names_files(4)));
laser1 = load(laser1_file);
imu3_file = strcat(directorio,char(names_files(5)));
imu3 = load(imu3_file);
laser2_file = strcat(directorio,char(names_files(6)));
laser2 = load(laser2_file);



%% Encoder

% Comprobación de representación del encoder.
```

```
% Carga del fichero de salida del mismo, pasado como parámetro de
esta
% función.
% El resultado se pasa a la función encargada de representar los
datos
% obtenidos, se puede consultar más abajo en este mismo fichero.
if encoder_file ~= 0
    % Se toma como tiempo límite de la simulación del último
instante de
    % tiempo de toma de datos por parte del encoder
    time_stop = encoder(end,5)*10^-3;
    figure
    hold on
    encoder_representacion(encoder,time_stop);
    hold off
end



%% IMUs

% Representación conjunta de las 3 IMUs
if length(imu1_file) ~= 0
    if length(imu2_file) ~= 0
        if length(imu3_file) ~= 0
            % Angular velocities
            figure, subplot(3,2,1), plot(imu1(:,1)*10^-
6,imu1(:,2),'b',imu3(:,1)*10^-6,imu3(:,2),'r',imu2(:,1)*10^-
6,imu2(:,2),'g'), title('Angular Velocity \omega_x'), xlabel('Time
(s)'), ylabel('\omega_x (deg/s)'),
legend('wheelset','bogie','carbody')
            xlim([0 time_stop])
            ylim([-5,5])
            subplot(3,2,3), plot(imu1(:,1)*10^-
6,imu1(:,3),'b',imu3(:,1)*10^-6,imu3(:,3),'r',imu2(:,1)*10^-
6,imu2(:,3),'g'), title('Angular Velocity \omega_y'), xlabel('Time
(s)'), ylabel('\omega_y (deg/s)'),
legend('wheelset','bogie','carbody')
            xlim([0 time_stop])
            ylim([-3,3])
            subplot(3,2,5), plot(imu1(:,1)*10^-
6,imu1(:,4),'b',imu3(:,1)*10^-6,imu3(:,4),'r',imu2(:,1)*10^-
6,imu2(:,4),'g'), title('Angular Velocity \omega_z'), xlabel('Time
(s)'), ylabel('\omega_z (deg/s)'),
legend('wheelset','bogie','carbody')
            xlim([0 time_stop])
            ylim([-2,2])

            % Cartesian accelerations
            % Se resta el offset del segundo valor tomado porque
el primero contiene cierto error en algunas ocasiones
            subplot(3,2,2), plot(imu1(:,1)*10^-6,imu1(:,5)-
imu1(2,5),'b',imu3(:,1)*10^-6,imu3(:,5)-
imu3(2,5),'r',imu2(:,1)*10^-6,imu2(:,5)-imu2(2,5),'g'), title('X-
acceleration a_x'), xlabel('Time (s)'), ylabel('a_x (g)'),
legend('wheelset','bogie','carbody')
```

```
            xlim([0 time_stop])
            ylim([-5,5])
            subplot(3,2,4), plot(imu1(:,1)*10^-6,imu1(:,6)-
imu1(2,6),'b',imu3(:,1)*10^-6,imu3(:,6),'r',imu2(:,1)*10^-
6,imu2(:,6)-imu2(2,6),'g'), title('Y-acceleration a_y'),
xlabel('Time (s)'), ylabel('a_y (g)'),
legend('wheelset','bogie','carbody')
            xlim([0 time_stop])
            ylim([-10,10])
            subplot(3,2,6), plot(imu1(:,1)*10^-6,imu1(:,7)-
imu1(2,7),'b',imu3(:,1)*10^-6,imu3(:,7)-
imu3(2,7),'r',imu2(:,1)*10^-6,imu2(:,7)-imu2(2,7),'g'), title('Z-
acceleration a_z'), xlabel('Time (s)'), ylabel('a_z (g)'),
legend('wheelset','bogie','carbody')
            xlim([0 time_stop])
            ylim([-20,20])
        end
    end
end



%% Lasers

% Comprobación de representación de la laser 1.
% Carga del fichero de salida del mismo, pasado como parámetro de
esta
% función.
% El resultado se pasa a la función encargada de representar los
datos
% obtenidos, se puede consultar más abajo en este mismo fichero.
if length(laser1_file) ~= 0
    figure, subplot(3,1,1), plot(laser1(:,1)*10^-
6,laser1(:,2),'b'), xlabel('Time (s)'), ylabel('Distance (mm)'),
title('Carbody distance laser')
    xlim([0 time_stop])
    ylim([80,87])
end

% Ídem para laser 2.
if length(laser2_file) ~= 0
    subplot(3,1,2), plot(laser2(:,1)*10^-6,laser2(:,2),'r'),
xlabel('Time(s)'), ylabel('Distance (mm)'), title('Bogie distance
laser')
    xlim([0 time_stop])
    ylim([10,16])
end

% Representación conjunta de los 2 lásers
% Comprobación de si se han pasado como parámetro los tres
ficheros correspondientes: "(imu1_file ~= 0)" devuelve un vector
fila de 1 de 1x27 en el caso de que se cumpla la condición.
% Se suman los valores de dichos vectores y ha de resultar 46 para
que se sepa que se han pasado los 2 ficheros y no un "0" en su
lugar como argumento.
if length(laser1_file) ~= 0
```

```
    if length(laser2_file) ~= 0
            % Distance lasers
            subplot(3,1,3), plot(laser1(:,1)*10^-6,laser1(:,2)-
laser1(2,2),'b',laser2(:,1)*10^-6,laser2(:,2)-laser2(2,2),'r'),
title('Relative Distance lasers \Deltaz (mm)'), xlabel('Time
(s)'), ylabel('\Deltaz (mm)'), legend('Carbody laser','Bogie
laser'),xlim([0 time_stop])
            ylim([-5,5])
    end
end

end


%% Función de representación encoder

function encoder_representacion(encoder,time_stop)

R = 0.0635;       % radio de la rueda
t_m = 0.008;      % tiempo de muestreo del sistemas de adquisición
8 ms, (seg)
f_adq = 1/t_m;    % frecuencia de muestreo del sistemas de
adquisición
f_nyq = f_adq/2; % tiempo de muestreo de Nyquist al doble del t_m
f_c = 1;
d_enc = 1500;     % dientes del encoder

time = encoder(:,5)*10^-3;     % instantes de tiempo de adquisición
de datos (seg)

% Se calcula la velocidad
vel = 2*pi*(encoder(:,4)/d_enc)/t_m*R;

% Se filtra la señal velocidad con un filtro de Butterworth
[b,a] = butter(4,f_c/f_nyq,'low');
OMega = filtfilt(b,a,vel);

% Se calcula el espacio recorrido
s = R*encoder(:,3)*2*pi/d_enc;

% Comprobación de que el espacio calculado entre instantes de
tiempo será
% correcto
s_acu = zeros(1,length(time)-1);
for i = 1:length(time)
    if i == 1
        s_acu(i) = t_m*vel(i);
    else
        s_acu(i) = s_acu(i-1) + t_m*vel(i);
    end
end

% Representación de la salida del encoder, velocidad y espacio
recorrido
subplot(2,1,1)
```

```
plot(time,vel,'r', time,OMega,'b')
title('Motor encoder')
legend('Real velocity','Filtered velocity')
xlabel('Time (s)'),ylabel('Velocity (m/s)'),xlim([0 time_stop])

subplot(2,1,2)
plot(time,s,'r', time,s_acu,'b')
legend('Real space','Calculated space')
xlabel('Time (s)'),ylabel('Space (m)'),xlim([0 time_stop])

end
```

2.2.2 Representación automática de datos obtenidos desde sensores con eliminación de datos espúreos – *auto_full_representation_filtered.m*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%% DESCRIPCIÓN: fichero que representa los datos obtenidos de los
sensores
%                mediante el sistema de adquisición de datos de
tren a
%                escala.
%                Sólo es necesario pasar el directorio donde se
encuentran
%                todos los ficheros ".csv".
%                DATO: todos los ficheros han de contener
información, de lo
%                contrario utilizar la función
"full_representation.m"
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%% AUTOR: María Dolores Sancho Martín
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%% FICHEROS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
% encoder_file: fichero de salida encoder del motor, (*_enc.csv),
6 columnas:
%                time: representa el triple de la diferencia entre
dos muestras consecutivas de la columna "elapsed time (ms)" (NOTA:
no es utilizado para las representaciones de este fichero)
%                omega_m (rpm): velocidad del motor en revoluciones
por minuto (NOTA: si es negativo la polarización a la que se ha
conectado el moto es correcta)
%                encoder_position:
%                position_change:
%                elapsed time (ms): instante de tiempo de toma de
datos.
% imu1_file: fichero de salida IMU 1 conectada al puerto serie 1
(SP1, 2 IMUs conectadas a él), (*_sp1imu.csv), 7 columnas:
%                Time (us): instante de tiempo de toma de datos.
```

```
%                Angular rate X (°/s): velocidad angular obtenida
entorno al eje x, obtenida en grados/segundo
%                Angular rate Y (°/s): velocidad angular obtenida
entorno al eje y, obtenida en grados/segundo
%                Angular rate Z (°/s): velocidad angular obtenida
entorno al eje z, obtenida en grados/segundo
%                Acceleration X (g): aceleración obtenida entorno
al eje x, obtenida en g
%                Acceleration Y (g): aceleración obtenida entorno
al eje y, obtenida en g
%                Acceleration Z (g): aceleración obtenida entorno
al eje z, obtenida en g
% imu2_file: fichero de salida IMU 2 conectada al puerto serie 1
(SP1, 2 IMUs conectadas a él), (*_sp1imu2.csv). Mismas columnas
que imu1_file.
% imu3_file: fichero de salida IMU 3 conectada al puerto serie 2
(SP2, 1 IMU conectada a él), (*_sp2imu.csv). Mismas columnas que
imu1_file.
% laser1_file: fichero de salida laser 1 conectado al puerto serie
1 (SP1, 1 laser conectado a él), (*_sp1laser.csv), 2 columnas:
%                Time (us): instante de tiempo de toma de datos.
%                distance (mm): distancia desde la salida del laser
hasta donde se proyecta el punto del mismo (NOTA: para correctas
mediciones el led del laser ha de estar en verde)
% laser2_file: fichero de salida laser 2 conectado al puerto serie
2 (SP2, 1 laser conectado a él), (*_sp2laser.csv). Mismas columnas
que laser1_file.
%
% DATO: Las líneas de título de los ficheros ha de estar
comentadas para
% introducirlos como parámetros, es decir, han de estar precedidas
del
% símbolo "%" (sin comillas)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

function auto_full_representation_filtered
close all


%% Load files

% Se asignan los archivos de salida de los distintos componentes
en formato .csv
directorio = 'ensayos_alamillo_28_9_17\2\';
d = dir(strcat(directorio,'*.csv'));
names_files = {d.name};

encoder_file = strcat(directorio,char(names_files(1)));
encoder = csvread(encoder_file,1,0);
imu1_file = strcat(directorio,char(names_files(2)));
imu1_peaks = load(imu1_file);
imu2_file = strcat(directorio,char(names_files(3)));
imu2_peaks = load(imu2_file);
laser1_file = strcat(directorio,char(names_files(4)));
laser1_peaks = load(laser1_file);
```

```
imu3_file = strcat(directorio,char(names_files(5)));
imu3_peaks = load(imu3_file);
laser2_file = strcat(directorio,char(names_files(6)));
laser2_peaks = load(laser2_file);


%% Se busca el instante de tiempo final más bajo de todos los
componentes en
% la toma de datos, pasados a segundos
stop_times = [encoder(end,5)*10^-3, imu1_peaks(end,1)*10^-6,
imu2_peaks(end,1)*10^-6, laser1_peaks(end,1)*10^-6,
imu3_peaks(end,1)*10^-6, laser2_peaks(end,1)*10^-6];
[time_stop,i] = min(stop_times);
% Para saber qué componente ha sido, se muestra por la línea de
comandos
% para nuestra información
componentes = [1,2,3,4,5,6];
fprintf('Componente que ha cortado la toma de datos el primero:
%i\n',componentes(i));



%% Procesado de ficheros, filtrado de picos
[imu1,imu2,imu3,laser1,laser2] =
peaks_filter(imu1_peaks,imu2_peaks,imu3_peaks,laser1_peaks,laser2_
peaks,time_stop);



%% Se busca el instante de tiempo final más bajo de todos los
componentes en
% la toma de datos, pasados a segundos
stop_times = [encoder(end,5)*10^-3, imu1(end,1)*10^-6,
imu2(end,1)*10^-6, laser1(end,1)*10^-6, imu3(end,1)*10^-6,
laser2(end,1)*10^-6];
[time_stop,i] = min(stop_times);
% Para saber qué componente ha sido, se muestra por la línea de
comandos
% para nuestra información
componentes = [1,2,3,4,5,6];
fprintf('Componente que ha cortado la toma de datos el primero:
%i\n',componentes(i));


%% Encoder

% Comprobación de representación del encoder.
% Carga del fichero de salida del mismo, pasado como parámetro de
esta
% función.
% El resultado se pasa a la función encargada de representar los
datos
% obtenidos, se puede consultar más abajo en este mismo fichero.
if encoder_file ~= 0
```

```
    % Se toma como tiempo límite de la simulación del último
instante de
    % tiempo de toma de datos por parte del encoder
    time_stop = encoder(end,5)*10^-3;
    figure
    hold on
    encoder_representacion(encoder,time_stop);
    hold off
end


%% IMUs

% Representación conjunta de las 3 IMUs
if length(imu1_file) ~= 0
    if length(imu2_file) ~= 0
        if length(imu3_file) ~= 0
            % Angular velocities
            figure, subplot(3,2,1), plot(imu1(:,1)*10^-
6,imu1(:,2)-imu1(2,2),'b',imu3(:,1)*10^-6,imu3(:,2)-
imu3(2,2),'r',imu2(:,1)*10^-6,imu2(:,2)-imu2(2,2),'g'),
title('Angular Velocity \omega_x'), xlabel('Time (s)'),
ylabel('\omega_x (deg/s)'), legend('Wheelset-IMU1','Bogie-
IMU3','Carbody-IMU2')
            xlim([0 time_stop])
            ylim([-5,5])
            subplot(3,2,3), plot(imu1(:,1)*10^-6,-(imu1(:,3)-
imu1(2,3)),'b',imu3(:,1)*10^-6,-(imu3(:,3)-
imu3(2,3)),'r',imu2(:,1)*10^-6,-(imu2(:,3)-imu2(2,3)),'g'),
title('Angular Velocity \omega_y'), xlabel('Time (s)'),
ylabel('\omega_y (deg/s)'), legend('Wheelset-IMU1','Bogie-
IMU3','Carbody-IMU2')
            xlim([0 time_stop])
            ylim([-2,2])
            subplot(3,2,5), plot(imu1(:,1)*10^-6,-(imu1(:,4)-
imu1(2,4)),'b',imu3(:,1)*10^-6,-(imu3(:,4)-
imu3(2,4)),'r',imu2(:,1)*10^-6,-(imu2(:,4)-imu2(2,4)),'g'),
title('Angular Velocity \omega_z'), xlabel('Time (s)'),
ylabel('\omega_z (deg/s)'), legend('Wheelset-IMU1','Bogie-
IMU3','Carbody-IMU2')
            xlim([0 time_stop])
            ylim([-2,2])

            % Cartesian accelerations
            % Se resta el offset del segundo valor tomado porque
el primero contiene cierto error en algunas ocasiones
            subplot(3,2,2), plot(imu1(:,1)*10^-6,imu1(:,5)-
imu1(2,5),'b',imu3(:,1)*10^-6,imu3(:,5)-
imu3(1,5),'r',imu2(:,1)*10^-6,imu2(:,5)-imu2(1,5),'g'), title('X-
acceleration a_x'), xlabel('Time (s)'), ylabel('a_x (g)'),
legend('Wheelset-IMU1','Bogie-IMU3','Carbody-IMU2')
            xlim([0 time_stop])
            ylim([-10,10])
            subplot(3,2,4), plot(imu1(:,1)*10^-6,-(imu1(:,6)-
imu1(2,6)),'b',imu3(:,1)*10^-6,-(imu3(:,6)-
imu3(1,6)),'r',imu2(:,1)*10^-6,-(imu2(:,6)-imu2(1,6)),'g'),
```

```
title('Y-acceleration a_y'), xlabel('Time (s)'), ylabel('a_y
(g)'), legend('Wheelset-IMU1','Bogie-IMU3','Carbody-IMU2')
            xlim([0 time_stop])
            ylim([-10,10])
            subplot(3,2,6), plot(imu1(:,1)*10^-6,-(imu1(:,7)-
imu1(2,7)),'b',imu3(:,1)*10^-6,-(imu3(:,7)-
imu3(1,7)),'r',imu2(:,1)*10^-6,-(imu2(:,7)-imu2(1,7)),'g'),
title('Z-acceleration a_z'), xlabel('Time (s)'), ylabel('a_z
(g)'), legend('Wheelset-IMU1','Bogie-IMU3','Carbody-IMU2')
            xlim([0 time_stop])
            ylim([-15,15])
        end
    end
end



%% Lasers

% Comprobación de representación de la laser 1.
% Carga del fichero de salida del mismo, pasado como parámetro de
esta
% función.
% El resultado se pasa a la función encargada de representar los
datos
% obtenidos, se puede consultar más abajo en este mismo fichero.
if length(laser1_file) ~= 0
    figure, subplot(3,1,1), plot(laser1(:,1)*10^-
6,laser1(:,2),'b'), xlabel('Time (s)'), ylabel('Distance (mm)'),
title('Carbody laser-LASER1')
    xlim([0 time_stop])
    ylim([80,87])
end

% Ídem para laser 2.
if length(laser2_file) ~= 0
    subplot(3,1,2), plot(laser2(:,1)*10^-6,laser2(:,2),'r'),
xlabel('Time(s)'), ylabel('Distance (mm)'), title('Bogie laser-
LASER2')
    xlim([0 time_stop])
    ylim([10,16])
end

% Representación conjunta de los 2 lásers
% Comprobación de si se han pasado como parámetro los tres
ficheros correspondientes: "(imu1_file ~= 0)" devuelve un vector
fila de 1 de 1x27 en el caso de que se cumpla la condición.
% Se suman los valores de dichos vectores y ha de resultar 46 para
que se sepa que se han pasado los 2 ficheros y no un "0" en su
lugar como argumento.
if length(laser1_file) ~= 0
    if length(laser2_file) ~= 0
            % Distance lasers
            subplot(3,1,3), plot(laser1(:,1)*10^-6,laser1(:,2)-
laser1(2,2),'b',laser2(:,1)*10^-6,laser2(:,2)-laser2(2,2),'r'),
title('Relative Distance lasers \Deltaz (mm)'), xlabel('Time
```

```
(s)'), ylabel('\Deltaz (mm)'), legend('Carbody laser-
LASER1','Bogie laser-LASER2'),xlim([0 time_stop])
            ylim([-5,5])
    end
end

end


%% Función de representación encoder

function encoder_representacion(encoder,time_stop)

R = 0.0635;       % radio de la rueda
t_m = 0.008;      % tiempo de muestreo del sistemas de adquisición
8 ms, (seg)
f_adq = 1/t_m;    % frecuencia de muestreo del sistemas de
adquisición
f_nyq = f_adq/2; % tiempo de muestreo de Nyquist al doble del t_m
f_c = 1;
d_enc = 1500;     % dientes del encoder

time = encoder(:,5)*10^-3;     % instantes de tiempo de adquisición
de datos (seg)

% Se calcula la velocidad
vel = 2*pi*(encoder(:,4)/d_enc)/t_m*R;

% Se filtra la señal velocidad con un filtro de Butterworth
[b,a] = butter(4,f_c/f_nyq,'low');
OMega = filtfilt(b,a,vel);

% Se calcula el espacio recorrido
s = R*encoder(:,3)*2*pi/d_enc;

% Comprobación de que el espacio calculado entre instantes de
tiempo será
% correcto
s_acu = zeros(1,length(time)-1);
for i = 1:length(time)
    if i == 1
        s_acu(i) = t_m*vel(i);
    else
        s_acu(i) = s_acu(i-1) + t_m*vel(i);
    end
end

% Representación de la salida del encoder, velocidad y espacio
recorrido
subplot(2,1,1)
plot(time,vel,'r', time,OMega,'b')
title('Motor encoder')
legend('Real velocity','Filtered velocity')
xlabel('Time (s)'),ylabel('Velocity (m/s)'),xlim([0 time_stop])
```

```
subplot(2,1,2)
plot(time,s,'r', time,s_acu,'b')
legend('Real space','Calculated space')
xlabel('Time (s)'),ylabel('Space (m)'),xlim([0 time_stop])

end



%% Función de procesado de ficheros, ausencia de picos

function [imu1,imu2,imu3,laser1,laser2] =
peaks_filter(imu1,imu2,imu3,laser1,laser2,time_stop)

k = 1;  % Para índices

%% Filtrados según el tiempo

valor_excep_time = time_stop + 0.05;

% Filtrado fichero IMU 1
i_imu1 = find(imu1(:,1)*10^-6 > valor_excep_time);

for i = 1:length(i_imu1)
    imu1(i_imu1(i),:) = [];
    i_imu1 = i_imu1-1;
end

% Puede ocurrir que algunas tomas de datos de los ficheros tengan
un valor
% de la toma de tiempo que no corresponda consecutivamente a las
anteriores
% pero no sea un valor disparatado, por lo tanto se revisa que
todo el
% fichero sea secuencial en la toma de datos
k=1;
while k < length(imu1)
    if (imu1(k,1) > imu1(k+1,1))
        imu1(k,:) = [];
    end
    k = k+1;
end



% Filtrado fichero IMU 2
i_imu2 = find(imu2(:,1)*10^-6 > valor_excep_time);

for i = 1:length(i_imu2)
    imu2(i_imu2(i),:) = [];
    i_imu2 = i_imu2-1;
end
```

```
% Puede ocurrir que algunas tomas de datos de los ficheros tengan
un valor
% de la toma de tiempo que no corresponda consecutivamente a las
anteriores
% pero no sea un valor disparatado, por lo tanto se revisa que
todo el
% fichero sea secuencial en la toma de datos
k=1;
while k < length(imu2)
    if (imu2(k,1) > imu2(k+1,1))
        imu2(k,:) = [];
    end
    k = k+1;
end


% Filtrado fichero IMU 3
i_imu3 = find(imu3(:,1)*10^-6 > valor_excep_time);

for i = 1:length(i_imu3)
    imu3(i_imu3(i),:) = [];
    i_imu3 = i_imu3-1;
end

% Puede ocurrir que algunas tomas de datos de los ficheros tengan
un valor
% de la toma de tiempo que no corresponda consecutivamente a las
anteriores
% pero no sea un valor disparatado, por lo tanto se revisa que
todo el
% fichero sea secuencial en la toma de datos
k=1;
while k < length(imu3)
    if (imu3(k,1) > imu3(k+1,1))
        imu3(k,:) = [];
    end
    k = k+1;
end


% Filtrado fichero LASER 1
i_laser1 = find(laser1(:,1)*10^-6 > valor_excep_time);

for i = 1:length(i_laser1)
    if (i_laser1(i) < 4)
        laser1(i_laser1(i),:) = [];
        i_laser1 = i_laser1-1;
    else
        laser1(i_laser1(i),:) = [];
        laser1(i_laser1(i)+1,:) = [];
        laser1(i_laser1(i)+2,:) = [];
        laser1(i_laser1(i)+3,:) = [];
        i_laser1 = i_laser1-4;
    end
end
```

```
% Puede ocurrir que algunas tomas de datos de los ficheros tengan
un valor
% de la toma de tiempo que no corresponda consecutivamente a las
anteriores
% pero no sea un valor disparatado, por lo tanto se revisa que
todo el
% fichero sea secuencial en la toma de datos
k=1;
while k < length(laser1)
    if (laser1(k,1) > laser1(k+1,1))

        laser1(k,:) = [];
    end
    k = k+1;
end



% Filtrado fichero LASER 2
i_laser2 = find(laser2(:,1)*10^-6 > valor_excep_time);

for i = 1:length(i_laser2)
    if (i_laser2(i) < 4)
        laser2(i_laser2(i),:) = [];
        i_laser2 = i_laser2-1;
    else
        laser2(i_laser2(i),:) = [];
        laser2(i_laser2(i)+1,:) = [];
        laser2(i_laser2(i)+2,:) = [];
        laser2(i_laser2(i)+3,:) = [];
        i_laser2 = i_laser2-4;
    end

end

% Puede ocurrir que algunas tomas de datos de los ficheros tengan
un valor
% de la toma de tiempo que no corresponda consecutivamente a las
anteriores
% pero no sea un valor disparatado, por lo tanto se revisa que
todo el
% fichero sea secuencial en la toma de datos
k=1;
while k < length(laser2)
    if (laser2(k,1) > laser2(k+1,1))
        laser2(k,:) = [];
    end
    k = k+1;
end



%% Filtrados según el valor de salida

valor_excepcional = 20;
```

```
% Filtrado fichero IMU 1
for k = 2:7
    i_imu1 = find(abs(imu1(:,k)) > valor_excepcional);

    for i = 1:length(i_imu1)
        imu1(i_imu1(i),:) = [];
        i_imu1 = i_imu1-1;
    end
end

% Filtrado fichero IMU 2
for k = 2:7
    i_imu2 = find(abs(imu2(:,k)) > valor_excepcional);

    for i = 1:length(i_imu2)
        imu2(i_imu2(i),:) = [];
        i_imu2 = i_imu2-1;
    end
end

% Filtrado fichero IMU 3
for k = 2:7
    i_imu3 = find(abs(imu3(:,k)) > valor_excepcional);

    for i = 1:length(i_imu3)
        imu3(i_imu3(i),:) = [];
        i_imu3 = i_imu3-1;
    end
end

% Filtrado fichero LASER 1
i_laser1 = find(abs(laser1(:,2))> valor_excepcional*100);

for i = 1:length(i_laser1)
    laser1(i_laser1(i),:) = [];
    i_laser1 = i_laser1-1;
end

% Filtrado fichero LASER 2
i_laser2 = find(abs(laser2(:,2))> valor_excepcional*100);

for i = 1:length(i_laser2)
    laser2(i_laser2(i),:) = [];
    i_laser2 = i_laser2-1;
end

end
```

2.3 Representación gráfica animada de datos obtenidos por las cámaras de auscultación

2.3.1 Representación gráfica animada de los vídeos obtenidos por las cámaras de auscultación, con especificación de ficheros de entrada – *video.m*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%% DESCRIPCIÓN: fichero que representa los vídeos obtenidos por
las cámaras
%                del sistema de adquisición de datos de tren a
escala.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%% AUTOR: María Dolores Sancho Martín
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%% FICHEROS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
% cam_left_file: fichero de salida procesado de cámara izquierda,
%                 (*_cam_left.sfer.txt).
%                 Cada fila consta de 600 columnas y está formada
por
%                 valores comprendidos entre [0 400], al ser
imágenes de
%                 400x600 píxeles. Cada valor de cada foto
representa el
%                 píxel situado en medio de la nube de píxeles de
dicha
%                 columna. Por lo tanto, la foto del perfil que se
desea ver
%                 estará formada por ceros al inicio y fin de dicha
fila con
%                 valores distintos de ceros por el centro (en el
caso de no
%                 haber ruido en la imagen)
% cam_right_file: fichero de salida procesado de cámara derecha,
%                  (*_cam_right.sfer.txt).
%                  Su contenido sigue la misma dinámica que fichero
anterior
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

function video(cam_left_file,cam_right_file)
close all

% Las imágenes son de 400x600 píxeles, se toman estos datos para
su
% representacion
alto = 400;
ancho = 600;
```

```
% La frecuencia de muestreo del vídeo es de 250 Hz
freq_m = 250;
t_m = 1/freq_m;

% Comprobación de existencia de ambos ficheros
if cam_left_file ~= 0
    % Cada fila es una imagen y cada columna la altura media (en
píxeles) de
    % los bits en blanco. Tiene 600 columnas por ser la imagen de
400x600 pixeles
    matriz_video_izq=load(cam_left_file);
    [num_fotos_izq,~] = size(matriz_video_izq);
    ancho_foto_izq = 1:ancho;
end

if cam_right_file ~= 0
    % Cada fila es una imagen y cada columna la altura media (en
píxeles) de
    % los bits en blanco. Tiene 600 columnas por ser la imagen de
400x600 pixeles
    matriz_video_der=load(cam_right_file);
    [num_fotos_der,~] = size(matriz_video_der);
    ancho_foto_der = 1:ancho;
end

if cam_left_file ~= 0
    if cam_right_file ~= 0
        figure;

        foto_izq = matriz_video_izq(1,:);
        foto_der = matriz_video_der(1,:);

        subplot(1,2,1)
        plot(ancho_foto_izq,foto_izq,'.');
        title('Cámara izquierda')
        set(gca,'yDir','reverse')
        axis([1 ancho 1 alto]);
        xlabel('píxeles'),ylabel('píxeles')

        subplot(1,2,2)
        plot(ancho_foto_der,foto_der,'.');
        title('Cámara derecha')
        set(gca,'yDir','reverse')
        axis([1 ancho 1 alto]);
        xlabel('píxeles'),ylabel('píxeles')


        for i = 2:num_fotos_izq
            foto_izq = matriz_video_izq(i,:);
            foto_der = matriz_video_der(i,:);

            pause(freq_m)
            subplot(1,2,1)
            plot(ancho_foto_izq,foto_izq,'.');
            set(gca,'yDir','reverse')
            axis([1 ancho 1 alto]);
```

```
            xlabel('píxeles'),ylabel('píxeles')
            title('Cámara izquierda')
            refresh(1)

            subplot(1,2,2)
            plot(ancho_foto_der,foto_der,'.');
            set(gca,'yDir','reverse')
            axis([1 ancho 1 alto]);
            xlabel('píxeles'),ylabel('píxeles')
            title('Cámara derecha')
            refresh(1)
        end
    elseif cam_right_file == 0
        foto_izq = matriz_video_izq(1,:);

        plot(ancho_foto_izq,foto_izq,'.');
        title('Cámara izquierda')
        set(gca,'yDir','reverse')
        axis([1 ancho 1 alto]);
        xlabel('píxeles'),ylabel('píxeles')

        for i = 2:num_fotos_izq
            foto_izq = matriz_video_izq(i,:);

            pause(freq_m)
            plot(ancho_foto_izq,foto_izq,'.');
            set(gca,'yDir','reverse')
            axis([1 ancho 1 alto]);
            xlabel('píxeles'),ylabel('píxeles')
            title('Cámara izquierda')
            refresh(1)
        end
    end

elseif cam_left_file == 0
    foto_der = matriz_video_der(1,:);

    plot(ancho_foto_der,foto_der,'.');
    title('Cámara izquierda')
    set(gca,'yDir','reverse')
    axis([1 ancho 1 alto]);
    xlabel('píxeles'),ylabel('píxeles')

    for i = 2:num_fotos_der
        foto_der = matriz_video_der(i,:);

        pause(freq_m)
        plot(ancho_foto_der,foto_der,'.');
        set(gca,'yDir','reverse')
        axis([1 ancho 1 alto]);
        xlabel('píxeles'),ylabel('píxeles')
        title('Cámara izquierda')
        refresh(1)
    end
end
end
```

2.3.2 Representación automática de gráfica animada de los vídeos obtenidos por las cámaras de auscultación – *auto_video.m*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%% DESCRIPCIÓN: fichero que representa los vídeos obtenidos por
las cámaras
%               del sistema de adquisición de datos de tren a
escala.
%               Sólo es necesario pasar el directorio donde se
encuentran
%               los dos ficheros ".txt".
%               DATO: todos los ficheros han de contener
información, de lo
%               contrario utilizar la función "video.m"
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%% AUTOR: María Dolores Sancho Martín
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%% FICHEROS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
% cam_left_file: fichero de salida procesado de cámara izquierda,
%                (*_cam_left.sfer.txt).
%                Cada fila consta de 600 columnas y está formada
por
%                valores comprendidos entre [0 400], al ser
imágenes de
%                400x600 píxeles. Cada valor de cada foto
representa el
%                píxel situado en medio de la nube de píxeles de
dicha
%                columna. Por lo tanto, la foto del perfil que se
desea ver
%                estará formada por ceros al inicio y fin de dicha
fila con
%                valores distintos de ceros por el centro (en el
caso de no
%                haber ruido en la imagen)
% cam_right_file: fichero de salida procesado de cámara derecha,
%                 (*_cam_right.sfer.txt).
%                 Su contenido sigue la misma dinámica que fichero
anterior
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

function auto_video
close all

%% Load files

% Se asignan los archivos de salida de los distintos componentes
en formato .csv
directorio = 'ensayos_alamillo_28_9_17\3\';
```

```
d = dir(strcat(directorio,'*.sfer.txt'));
names_files = {d.name};

cam_left_file = strcat(directorio,char(names_files(1)));
cam_right_file = strcat(directorio,char(names_files(2)));

% Las imágenes son de 400x600 píxeles, se toman estos datos para
su
% representacion
alto = 400;
ancho = 600;

% La frecuencia de muestreo del vídeo es de 250 Hz
freq_m = 250;
t_m = 1/freq_m;

% Comprobación de existencia de ambos ficheros
if length(cam_left_file) ~= 0
    % Cada fila es una imagen y cada columna la altura media (en
píxeles) de
    % los bits en blanco. Tiene 600 columnas por ser la imagen de
400x600 pixeles
    matriz_video_izq=load(cam_left_file);
    [num_fotos_izq,~] = size(matriz_video_izq);
    ancho_foto_izq = 1:ancho;
end

if length(cam_right_file) ~= 0
    % Cada fila es una imagen y cada columna la altura media (en
píxeles) de
    % los bits en blanco. Tiene 600 columnas por ser la imagen de
400x600 pixeles
    matriz_video_der=load(cam_right_file);
    [num_fotos_der,~] = size(matriz_video_der);
    ancho_foto_der = 1:ancho;
end

if length(cam_left_file) ~= 0
    if length(cam_right_file) ~= 0
        figure('units','normalized','outerposition',[0 0 1 1])

        foto_izq = matriz_video_izq(1,:);
        foto_der = matriz_video_der(1,:);

        subplot(1,2,1)
        plot(ancho_foto_izq,foto_izq,'.');
        title('Cámara izquierda')
        set(gca,'yDir','reverse')
        axis([1 ancho 1 alto]);
        xlabel('píxeles'),ylabel('píxeles')
        daspect([1,1,1])

        subplot(1,2,2)
        plot(ancho_foto_der,foto_der,'.');
        title('Cámara derecha')
        set(gca,'yDir','reverse')
```

```
        axis([1 ancho 1 alto]);
        xlabel('píxeles'),ylabel('píxeles')
        daspect([1,1,1])


        for i = 2:num_fotos_izq
            foto_izq = matriz_video_izq(i,:);
            foto_der = matriz_video_der(i,:);

            pause(t_m)
            subplot(1,2,1)
            plot(ancho_foto_izq,foto_izq,'.');
            set(gca,'yDir','reverse')
            axis([1 ancho 1 alto]);
            xlabel('píxeles'),ylabel('píxeles')
            title('Cámara izquierda')
            daspect([1,1,1])
            refresh(1)

            subplot(1,2,2)
            plot(ancho_foto_der,foto_der,'.');
            set(gca,'yDir','reverse')
            axis([1 ancho 1 alto]);
            xlabel('píxeles'),ylabel('píxeles')
            title('Cámara derecha')
            daspect([1,1,1])
            refresh(1)
        end

    elseif length(cam_right_file) == 0
        foto_izq = matriz_video_izq(1,:);

        plot(ancho_foto_izq,foto_izq,'.');
        title('Cámara izquierda')
        set(gca,'yDir','reverse')
        axis([1 ancho 1 alto]);
        xlabel('píxeles'),ylabel('píxeles')

        for i = 2:num_fotos_izq
            foto_izq = matriz_video_izq(i,:);

            pause(t_m)
            plot(ancho_foto_izq,foto_izq,'.');
            set(gca,'yDir','reverse')
            axis([1 ancho 1 alto]);
            xlabel('píxeles'),ylabel('píxeles')
            title('Cámara izquierda')
            refresh(1)
        end
    end

elseif length(cam_left_file) == 0
    foto_der = matriz_video_der(1,:);

    plot(ancho_foto_der,foto_der,'.');
```

```
    title('Cámara izquierda')
    set(gca,'yDir','reverse')
    axis([1 ancho 1 alto]);
    xlabel('píxeles'),ylabel('píxeles')

    for i = 2:num_fotos_der
        foto_der = matriz_video_der(i,:);

        pause(t_m)
        plot(ancho_foto_der,foto_der,'.');
        set(gca,'yDir','reverse')
        axis([1 ancho 1 alto]);
        xlabel('píxeles'),ylabel('píxeles')
        title('Cámara izquierda')
        refresh(1)
    end
end

end
```

3 ANEXOS: GUÍA PUESTA EN MARCHA, ADQUISICIÓN Y PROCESADO DE DATOS

Puesta en marcha, adquisición y procesado de datos Sistema de Adquisición de Datos de tren a escala

María Dolores Sancho Martín

Puesta en marcha

1. Comprobar que todas las conexiones están correctamente en el tren mirando esquema de Anexo 4 del proyecto *Implementación y análisis de un sistema de adquisición de datos para un vehículo ferroviario a escala* de *María Dolores Sancho*, incluyendo el router (para el caso de control remoto).
2. Alimentar el tren mediante baterías o conectando la regleta a la corriente eléctrica (para el caso de pruebas en laboratorio).
3. Encender el ordenador del tren mediante la pestaña lateral y la bombilla parpadeará entre ámbar y verde. Cuando se quede fija en verde significará que se ha encendido.
4. Hay dos opciones para la visualización:
   a. Conectando pantalla, teclado y ratón al PC del tren. Pasar al paso 5.
   b. Mediante remoto haciendo uso del router. Para este caso seguir los siguientes pasos.
      i. Es necesario descargar en un ordenador pórtatil, (o aquel ordenador que tenga la capacidad de captar señales wifi), la aplicación "TeamViewer" e instalarla.
      ii. Se realiza la conexión a la red wifi con la ID: "esfucon", y como contraseña: "esfucon16".
      iii. Abrimos la aplicación "TeamViewer" y se introduce como ID de asociado: "192.168.1.40", estará seleccionada la opción "Control remoto" y se clicka en "Conectar con asociado". Como contraseña: "esfucon16"
5. Se muestra el escritorio del PC del tren y con pestañas, las cuales se han de cerrar. (Si se conecta en remoto y no se visualiza correctamente, modificar la resolución en "Ver"->"Resolución de pantalla")
6. Para comprobar la alineación de las cámaras, abrimos el programa "xiViewer" y nos deja seleccionar una de las dos cámaras. Para visualizar ambas, abrir de nuevo "xiViewer" y seleccionar la otra cámara.
   NOTA: en el caso que no se visualice alguna cámara reiniciar el PC del tren con los USBs de las cámaras desenchufados y conectarlos cuando el PC esté encendido de nuevo. Además, visualizar las cámaras una a una, no las dos a la vez. Se enfocan manualmente las cámaras y, si se desea ver exclusivamente el lasér oscureciendo el fondo, se le da al botón derecho del ratón. En la ventana se quita la selección de "AEAG" y se puede variar la barra de "Exposure".
7. Se cierra xiViewer, ya que no se puede emplear este programa a la vez que la adquisición de datos.
8. Abrimos el programa "Qt Creator". Seleccionar, dentro de "Recent Projects", "tep7280_daqsysmicro". En la nueva ventana se clicka en Run (botón en la parte inferior izquierda de la pantalla, verde con forma de punta de flecha) o bien a Crtl+R, apareciéndonos una nueva ventana.

Adquisición de datos

9. La ventana que se presenta controla los diferentes componentes encargados de la adquisición de datos y controla el funcionamiento del motor.
   Lo que se utiliza son los recuadros titulados:
   a. MOTOR DRIVE
   b. Microcontroller acquiring on Serial Port 1
   c. Microcontroller acquiring on Serial Port 2
   d. Sync cameras
   e. Right camera
   f. Left camera

   La activación de cada uno de ellos se explica en los siguientes pasos.
   i. MOTOR DRIVE:
      i. En la barra correspondiente a "s" se selecciona el ángulo de la pendiente con la que se desea que el motor adquiera paulatinamente velocidad hasta que llegue a la que se desea, está en %. Normalmente, "s = 25", ha de ser tal que las ruedas del tren no patinen al iniciar la marcha.
      ii. La velocidad se encuentra en %, y se escribe directamente en el recuadro "vmax". Se puede poner 100% o lo que se desee, 0% para que se quede parado el motor.
      iii. Ya no se modifica nada más de este recuadro, los botones de "run & save", "run" o "stop" no se utilizan. Sólo destacar que la información que se adquiera del encoder del motor se guardará en el fichero indicado: "*_enc.csv", donde * es un distintivo de cada fichero generado en función de la fecha y hora en la que se realice.
   j. Microcontroller acquiring on Serial Port 1:
      i. "Activar/desactivar": seleccionar para activar la adquisición de datos por parte de la Mbed que tiene conectada dos IMUs y un láser, todo ello al puerto serie 1 (SP1). Aparecerá CONNECTED o DISCONNECTED" en "Estado".
      ii. El puerto que suele reconocer el PC es COM25, (comprobar si es ese al desenchufar y enchufar la Mbed que contiene dos IMUs en "Inicio"->"Dispositivos e impresoras").
      iii. Los ficheros que devuelve es uno por cada IMU y otro perteneciente a la salida del láser de medición de distancias.
   k. Microcontroller acquiring on Serial Port 2:
      i. "Activar/desactivar": seleccionar para activar la adquisición de datos por parte de la Mbed que tiene conectada una IMU y un láser, todo ello al puerto serie 2 (SP2). Aparecerá CONNECTED o DISCONNECTED" en "Estado".
      ii. El puerto que suele reconocer el PC es COM24, (comprobar si es ese al desenchufar y enchufar la Mbed que contiene una imu en "Inicio"->"Dispositivos e impresoras").
      iii. Los ficheros que devuelve es uno por cada imu y otro perteneciente a la salida del láser de medición de distancias.

l. Sync cameras:
    i. "Activar/desactivar": seleccionar para activar la adquisición de datos por parte de la Mbed que envía un pulso para informar a las cámaras de que empiecen a grabar.
    ii. El puerto que suele reconocer el PC es COM14, (comprobar si es ese al desenchufar y enchufar la Mbed en "Inicio"->"Dispositivos e impresoras").
    iii. No devuelve fichero ya que no es relevante.
m. Right camera:
    i. "Activar/desactivar": seleccionar para activar la adquisición de datos por parte de la cámara. Aparecerá "CONNECTED" o "DISCONNECTED" en "Estado".
    ii. Se asigna automáticamente el número de serie de la cámara, "SN: 07431751".
    iii. El fichero que devuelve está compuesto por líneas representantes de las fotos del vídeo grabado por la cámra, las cuales tienen 600 columnas, al ser las fotos de 600x400 píxeles.
n. Left camera:
    i. "Activar/desactivar": seleccionar para activar la adquisición de datos por parte de la cámara. Aparecerá "CONNECTED" o "DISCONNECTED" en "Estado".
    ii. Se asigna automáticamente el número de serie de la cámara.
    iii. El fichero que devuelve está compuesto por líneas representantes de las fotos del vídeo grabado por la cámara, las cuales tienen 600 columnas, al ser las fotos de 600x400 píxeles.
o. Una vez activados los elementos que se deseen, se da al boton Start adquisition (botón azul con punta de flecha blanca, en la parte superior de la pantalla).
p. Cuando se desee parar la ejecución, se pulsa el botón Stop adquisition (botón rojo con cuadrado blanco, en la parte superior de la pantalla).

Procesamiento de datos

10. Todos los ficheros de las adquisiciones se guardan dentro del PC en el directorio: "C:\Users\Virtualmech\Documents\Tep 7280\tep7280_daqsysmicro-copia_results"
11. Si se desean visualizar lo datos de las cámaras de forma rápida, se abre el ejecutable "adqCam". En la ventana se escribe "3" y se presiona ENTER. Tras ello, se escribe el PATH incluyendo el nombre del fichero de salida de la cámara (es válido pegar con el botón derecho).
12. Para procesarlos, dependiendo de si nos hemos conectado directamente al PC del tren o bien por remoto:
    a. Directamente al PC del tren: se copian los ficheros deseados en un pen (que se conecta a un USB del PC) y se pasan a un ordenador que tenga instalado el programa Matlab.
    b. Remoto desde portátil: se copian los ficheros desde el TeamViewer y se pegan en nuestro ordenador en el directorio deseado.
13. Se ejecutará el programa creado para la representación de los distintos datos: "auto_full_representation", para ello es necesario previamente entrar en él y pegar el PATH de los ficheros en la variable "directorio".

14. Para la visualización en modo vídeo en Matlab de los dicheros devueltos de las cámaras, seguir los siguientes pasos:
    a. Copiar el ejecutable "BinPlainConverter.exe" y pegarlo en el directorio donde se encuentran los ficheros "*.sfer" devueltos por una de las cámaras.
    b. Abrir el ejecutable "BinPlainConverter.exe" y escribir el nombre completo con su extensión de uno de los ficheros "*.sfer". Generará un .txt con dicha información en el directorio actual. Se cerrará el ejecutable una vez realizada la operación, así que hay que abrirlo de nuevo para hacer lo mismo con el otro fichero.
15. Se ejecutará el programa creado para la representación en Matlab de los vídeos generados por las cámaras: "auto_video", para ello es necesario previamente entrar en él y pegar el PATH de los ficheros "*.sfer" en la variable "directorio".
    DATO: para distintas representaciones y procesamiento de los datos mirar las descripciones de los ficheros de Matlab para tal efecto.

4 ANEXOS: MAPA CONEXIONADO PLACA